SINCLAIR

The Premier Magazine

QL

for Sinclair QL Users

EVERY MONTH

£1.75

NOVEMBER 1990

DIY TOOLKIT

Faster disk check and access

One Man's System

One Man's System

BLOOD GLUCOSE RECORDS

THE PROGS

Three dimensional Pie

PROGRAMMING IN C

The second part

PRINTING FROM FLASHBACK

SuperBasic

ARCADE GAMES IN SUPERBASIC

WORLD

9 770951 933009 11

SINCLAIR QL WORLD

Editor
Helen Armstrong

Production Controller
Jayne Penfold

Designer
Jeff Gurney

Advertising Sales
Jason Newman

Magazine Services
Sheila Baker

Advertising Production
Michelle Evans

Group Advertising Manager
Richard Vaughan

Group Editor
John Taylor

Publishing Director
Ray Lewis

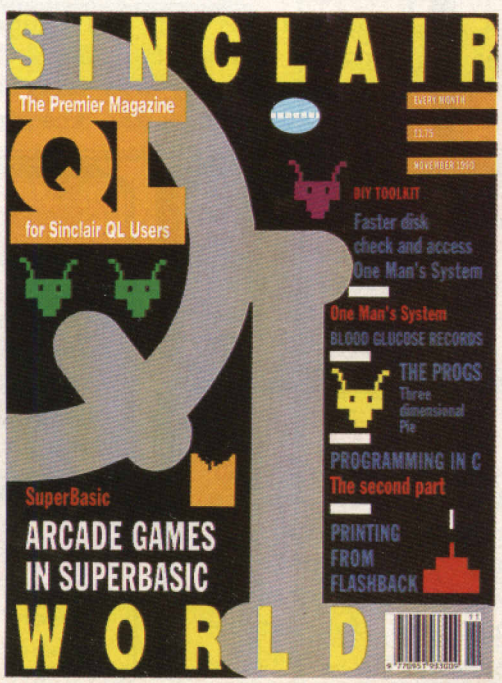
Managing Director
Peter Welham

Sinclair QL World
Panini House
116-120 Goswell Road
London EC1V 7QD
Telephone 071-490 7161
ISSN 026806X

Unfortunately, we are no longer able to answer enquiries made by telephone. If you have any comments or difficulties, please write the The Editor, Open Channel, Trouble Shooter, or Psion Solutions. We will do our best to deal with your problem in the magazine, though we cannot guarantee individual replies.
Back issues are available from the publisher price £2 U.K., £2.75 Europe. Overseas rates on request.
Published by Maxwell Specialist Magazines, A Division of MCPC Ltd., Sinclair QL World is distributed by SM Distribution.
Tel: 081-677 8111. Subscription information from: MSM Subscription Dept, Lazahold Ltd., PO Box 10, Roper St, Pallion Ind. Est., Sunderland SR4 4SN.
£21.00 U.K., £24.70 Europe, MiddleEast £25.80, Far East £27.60, Rest of World £26.20, U.S.A. $45.00.
Airmail rates available on request.
Typesetting by Ford Graphics, 8-10 Whitsbury Road, Fordingbridge, Hampshire. SP6 1BR. Tel: (0425) 655657
Printing by Cradley Print.
Sinclair QL World is published on the third Thursday preceding cover date.

CONTENTS

NOVEMBER 1990



NEXT MONTH

THE REALLY USEFUL ARCHIVE BOOKLET

To celebrate and abet our fresh assault on news-stands and newsagents, QL World carries an EXTRA free booklet about Archive on the cover next month.

QL SCENE

NEW FROM WALES

Dilwyn Jones Computing has supplied QL World with information on five programs which the new company is now marketing.

Dilwyn Jones is already well known to established QL users as the author of *Page Designer 2*, published by Sector Software, and other programs including some in the public domain. He hopes to establish himself as a QL supplier after some success with his programs through private advertising.

QL Basic Reporter is a programming utility for programmers in SuperBasic and compiled Basic. It has a variety of listing and reporting functions to list variable names, types, procedure and function names and lines, machine code extensions and execution addresses, and others. All lists can be sorted into alphabetical order and printed to the screen or any QL device. The program is useful among other things for finding typos and variable name clashes in listings. The program runs on an unexpanded QL and is designed to be easy to use. Basic Reporter is £10 including manual and EEC postage.

QL Vision Mixer is a screen sequencer for display and advertising. Screens and graphics can be loaded from just about any QL source, and then displayed at random or from a menu sequence, with speed and pauses user definable, including key-stepped displays. QL Vision Mixer needs at least 256K of expansion, and costs £10 including EEC postage.

QL Quick Posters is an aid to laying out text-only display sheets quickly. It exploits many little-used printer facilities. Designed for use with 24-pin printers – printers capable of doing enlarged text and with a choice of fonts are the most likely candidates. The program has no fonts of its own, but will improve the quality of large-size text, add borders, preview on-screen, load and save, and has a freely reconfigurable printer driver. A sample printout from a Star LC24-10 is available from the publisher with an sae. Quick Posters costs £10 including EEC postage.

QL Wordscheck takes any Ascii or Quill _doc file and provides a total word-count and the number of different words used. Output can be sent to screen, printer or file. The output can be given a number of user-definable variables such as a count of words over a certain length.

Wordscheck runs on an unexpanded QL and can multitask. The price is £6 including EEC postage.

QL Home Budget is an undated version of the already-established finance, income tax and capital gains tax calculator. Personal expenditure and bills can be entered as a number of accounts with itemised payments. The tax calculator assists in calculating personal income tax for any year from 1883/4 onwards, and includes separate taxation for married couples. Changes in taxation rules can be user-updated. It also covers "indexed" asset costs for capital gains tax purposes.

The current version addresses points raised in the May 1990 QL World review, plus extended error-trapping and multitasking using EXEC and CTRL C.

Home Budget costs £20 including EEC postage. It is QLiberated and runs on an unexpanded QL.

Postage outside the EEC area will be added at cost. All programs are available on mdv, 3.5 in disk or 5.25 in disk, and have printed manuals with the exception of Quick Poster, which includes instructions as a Quill_doc file.

Enquiries to **Dilwyn Jones Computing, 41 Bro Emrys, Tal-y-bont, Bangor, Gwynedd LL57 3YT.**

Assistance from Archive Author

Two mistakes have come to light in the *Archive Power* series, for which I must accept responsibility. The first is an error in the June 1990 listing, p34, towards the end of proc NOTEPAD. The line commencing 'if OPTION=4: error PAGETEMP: ....' should read 'if OPTION=4: error PAGEPRINT: ....' Then in the August 1990 issue, in the text on page 37, there is a list of procedures to be included in the calendar program. This includes the non-existant procedure GETSEDIT . It should have been GETSINPUT. Sorry to anyone frustrated by this.

Some readers have also found difficulty getting the program to run at all, due to confusion over how to save a program in *object* format. If you save a program in the normal way, by typing SAVE "MYPROG", it will be stored as a text file, with the name "MYPROG_PRG", and if it is a long program it will take an age to load even from disk. However, Psion also allow saving in a 'tokenised' format, with the line SAVE OBJECT "MYPROG". The file will then have the name "MYPROG_PRO". The main programs (NOTE PAD, CALENDAR and TODOLIST) should all be saved with this object form of the save command, to make use of the rapid loading it provides. If you have not done this, you will get an error message when July's initialisation program is run, saying 'cannot open file'. The solution is simple. From the Archive command line, simply enter LOAD 'NOTEPAD' followed by SAVE OBJECT 'NOTEPAD' and then try running the Desktop program.

Robin Stevenson

Show Success

The All Formats Computer Fair will be taking place again at the New Horticultural Hall, Greycoat St., Westminster, London on Saturday 1 September and Saturday 4 November.

A report reached QL World and its sister magazines that one trader reached the July fair with several thousand cassettes for various computers, ex-stock of from a retailer which had closed down. The haul was said to include around 2000 mdvs of unidentified QL software, which was "all gone by 2am". Windfalls like this are of course unpredictable, but QL visitors must have thought it was Christmas.

For advance £3.00 tickets and information, contact **Mike Hayes, 9 Midgrove, Delph, Oldham OB1 5EJ, tel. 0457 875229.**

OPEN CHANNEL

Open Channel is where you have the opportunity to voice your opinions in *Sinclair QL World*. Whether you want to ask for help with a technical problem, provide somebody with the answer, or just sound off about something which bothers you, write to: Open Channel, Sinclair QL World, 116/120 Goswell Road, London EC1v 7QD.

Thanks

I recently sent my *Minerva* rom for upgrade. The postman ensured that the rom pcb stapled itself and the rest of the package contents together. QView, however, replaced the pcb at no extra charge, and returned my upgrade eight days later.

Having installed Minerva 1.82, I found that Supercharged programs would no longer run. Phone calls to Stuart McKnight of QView and Freddy Vachha of Digital Precision ensured. Stuart explained the changes to Minerva which makes it less tolerant of programs which circumvent Qdos, and Freddy explained to me the reasons behind the incompatibility and my options. He then offered an upgrade to *Turbo* with a trade-in allowance of £30 for *Supercharge*. This was on top of the generous package discounts.

Both Stuart and Freddy I found to be helpful, informed, enthusiastic and shared my loathing of the IBM PC. At work I am forced to use an XT clone with *PC Write* and *Perfect Writer*, where reformatting of a large document, line by line, could take all day. This is something that the QL running DP's *The Editor* would usually achieve in about 30 minutes; five minutes, if I could fit Miracle Systems' 68020 prototype board.

My thanks to Tony Tebby, Jan Jones, Quanta, QView, Digital Precision, Miracle Systems and *QL World* for making the QL what it is.

B R Seidel
Manchester

Typing

Further to a reader query about *Dir_To_Archive_Bas*, I have suggested that he check his typing especially in procedures read_tempdir in lines 580 and 680 and open_file in lines 370 to 440, as these lines may have been mistyped. It is critical that there are no omissions of semicolons, backslashes, quote marks or ampersands as these are essential to the running of the routine. I have checked the listing as published and it is right.

Terry McKnight
Walkden, Manchester

Errata

Thank you for publishing my program *Chemistry* in the September issue. I had given up all hope of seeing it in print. However, on checking the listing, I found an omission in line 750.

After the last character on that line there should be two spaces and a "close of string" indicator ('). The line should look something like this:

```
750 PAPER 4,2: INK 7: AT 2,74: PRINT ' He ': AT 3,74: PRINT ' 2 '
```

This was probably spotted by many readers as the line would not be accepted by the QL without the final character, but I would like to point it out anyway.

A similar gremlin also attacked lines 1750 and 1770.

Also, if anyone has compiled the rest of the program, then the absence of a "Quit" option forces the user to reset the QL to escape. This may not be the case when using a front-end utility (I still don't own one) but it is an obvious omission. Adding the following lines provides the necessary function:

Change line 1840 by altering opt$>'3' to opt$>'4'

Add:

```
1805 PRINT: PRINT '4) Quit CHEMISTRY'
1875 IF opt$='4' THEN : quit
1882 DEFine PROCedure quit
1884 CLS: AT 10, 10: PRINT 'Bye'
1886 PAUSE 300: CLS: STOP
1888 END DEFine
```

After RENUMbering, the last line should be 3040.

The program still runs fine without this little addition, and runs and compiles successfully with it, even if parts of the code make some programmers wince!

On a different note, I would just like to add all the usual stuff about the magazine being wonderful, keep up the good work, will anyone produce a 'proper' QL Mark II and support for us all now?

I also hope that this letter meets at least 50% of the wishes which Bryan Davies expressed in September's *Troubleshooter*. It hasn't been checked by anything other than myself. There is only one deliberate spelling mistache; anything else must be Monday Morning Blues at the typesetting office.

Ian Thompson
Ripon
Yorks

Editor's comment: Probably Friday Afternoon Blues.

Mouse

I have heard on and off about the lack of a suitable mouse for the QL. I find this surprising as I have a mouse from Jochen Merz software here in Germany. It works perfectly, and I have had good service from Jochen; I see he now advertises in *QL World*. I installed the interface myself and if I can do it, it cannot be very difficult. I must say that I sweated a bit but it turned out well. The main complication came from the Schon interface, which normally interferes with it, but I put in a couple of spacers to lift it up.

Peter Lund
Munster, West Germany

Editor's notebook

The good news is that next month, with the December issue, our publishers will be investing the time and money that we have long been hoping for in distributing Sinclair QL World more widely to the news trade in the UK. This has been made possible by a change of distribution arrangements and our more settled lifestyle since we came to MSM.

So now is the time to tell your friends that they can get QL World from good newsagents, and that they can support the QL by buying their own copy!

To encourage our regular readers, as well as reach out to the many users who either don't know that QL World exists or can't afford to subscribe, we will have extra pages in the form of a free giveaway booklet on our December and January covers, and a return to the much-loved 'QL Old Master' style of cover artwork.

Look out for us in December, and if you don't see QL World on your newsagents' shelves – ask them why not.

Footnote: DIY Toolkit regrets that it now has to ask readers ordering batches on MDV to send sufficient formatted mdvs with their order, as they now not able to buy and supply mdvs economically within their prices.

QL SCENE

Budget

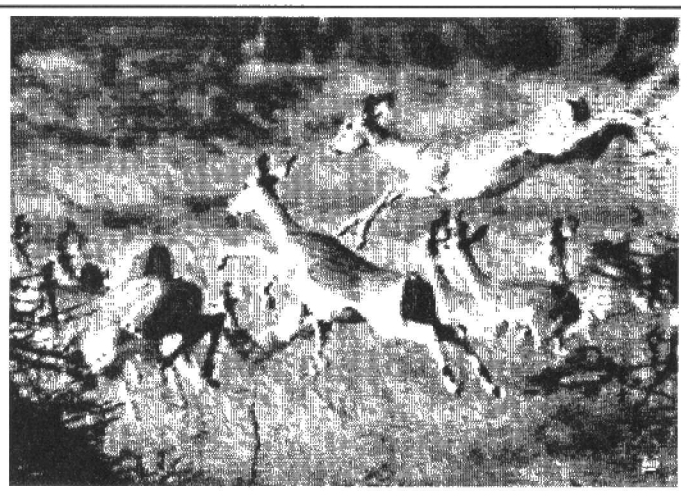
Joe Haftke, author of *Home Budget*, has terminated a sales and marketing agreement with QL supplier PDQL as of 29 July 1990, following the issue of the required legal notices, and requested return of all material. The marketing agreement with PDQL retained copyright to the author. At the time of reporting Mr. Hatfke has received no response from PDQL.

PDQL has been under a shadow in recent months following a spate of complaints from customers, including reports of non-delivery of goods after payment, repeated failure to meet delivery promises, and doubled charges on credit card accounts.

Mr Haftke dates his growing dissatisfaction "with PDQL's performance, or the lack of it" from some considerable time before a review of Home Budget appeared in the May 1990 issue of QL World.

An updated and improved release of Home Budget is now available from Dilwyn Jones Computing, who now have sales and marketing rights. "The contrast is a pleasure to experience", writes Mr Haftke of the new agreement.

Enquiries to **Dilwyn Jones Computing, 4 Bro Emrys, Tal-y-Bont, Bangor, Gwynedd LL57 3YT. Tel. 0248 354023.**



Digitiser

CL Systems proprietor C L Lang is proposing to release a new QL digitiser on the market in mid-October.

The CQV1 real time video digitiser is designed to capture and freeze frames from a video recorder, camera or other PAL-compatible video source in real time (about 20 milliseconds). The frames thus captured can be stored, printed or loaded into desktop publishing programs. Some of the potential uses projected by the manufacturer include slow frame recording, video graphics to printer, and capturing images of 3-D objects for desktop publishing.

The specifications include one odd or even frame in 20ms, display rate 5 frames per second in quarter-screen mode, 2.7 fps in full screen mode, resolution 256 by 256 pixels in eight grey levels, black level adjustment, display merge (OR, XOR), invert and negative, full quarter or 4x quarter displays. The CQV1 will run on mdv, 3.5 in or 5.25 in disk, on an unexpanded QL via the rom port.

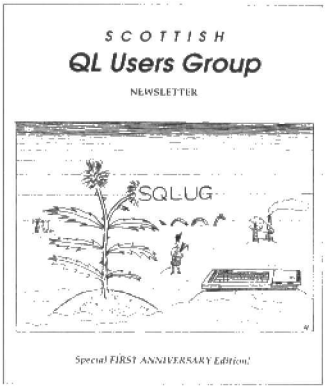
The projected price is £120 plus £3 p&p. For further information, samples or specification sheet contact **403 Chapter Road, Dollis Hill, London NW2 5NG, Tel. 081 459 1351.** QL World hopes to try out and review the CQV1 as soon as production models are available.

Demo

DataDesign by Progs - Van der Auwera is now available as a demo version for £3 or Belgian Francs 180 (cash or Visa). This version is said to have all the facilities of the operational package, except the Save commands. Potential users can experiment with the demo on their own machines before deciding whether or not to buy the program.

Enquiries to **Progs - Van der Auwera, Haachtstraat 92, 3020 Veltem, Belgium.**

SQLUG celebrates



The Scottish QL Users Group (SQLUG) is entering its second year with a celebratory First Anniversary issue of its newsletter, appropriately known as the Special Edition. The anniversary issue's main feature is a series of historical articles about the QL.

The Group meets monthly, usually for a software or hardware demonstration and discussion. The September meeting, for instance, is scheduled to include a video digitising session and an in-depth look at PC Conqueror.

The annual membership fee is currently £6 renewable in August, and pro rata at any other time of year. Enquiries to **SQLUG, Alan Pemberton, 65 Lingerwood Road, Newtongrange, Midlothian EH22 4QQ**

C connections corrected

Following last month's backslash corrections to *Programming in C*, we have some more details to correct, mainly involving confusion between the uses of hyphens and underscores.

In **Fig. 2,** line 3: Printf ("My first C program \ndisplays two lines");

On page 23, column 1:

Line 34 device flpl_ use:
Line 38 flp2_test -p -m (ENTER)
Line 41 code is in a file named flp2_test_c, that it ...

On page 23, column 2:

Line 5 _c doesn't need to be ...
Line 8 named flp2_test_obj
Line 11 exec_w flp1_cg (ENTER)
Line 19 file named flp2_test_obj

On page 23, column 3, line 2:

exec_w flp2_text_exe (ENTER)

Fig. 3:

Line 1 flp2_test_c→ flp2_test_obj → flp2_test_exe

And finally, a missing line on page 22, column 2, line 8:

... compatible) environment, a Unix environment and a VME environment (the operating system ...)

PRINTING FROM FLASHBACK

Bryan Davies offers assistance with printing databases directly from Flashback, or via text[87].

An enquiry from a reader about printing from *Flashback* and text[87] made the point which has often been heard in the past — users have problems with printing, from any program. What may appear a simple procedure to an experienced user, can be unknown, or very complicated, to someone who does not spend all day, every day using the QL. Here is what was suggested to this reader.

You can print databases directly from FlashBack, rather than going through another program. As FlashBack is flexible, it should normally be possible to do any editing required within it. However, there is no great difficulty writing the file out in a form which can be edited in text[87], and subsequently printed out there.

Using FlashBack alone, the <CTRL-P> keying enables the sending of a database direct to a printer. If you key <CTRL-P>, you will be offered SER1 as the default device to print to and most users should accept that. You can choose PAR if using a Sandy SuperQ board with parallel printer port.

The first illustration shows one record of a typical name-and-address database, after <CTRL-P> has been keyed; the program is waiting to be told whether or not the printer is connected (and switched on). There is one step to take before using the Print command, however — decide whether or not you want to print the whole file. To print only part of the file, use the Group command first, to select those records you want printed.

You can also use the Xclude command after Group, to remove odd records that meet the Group criteria but are not wanted in the printout. For example, to produce the printout in the second illustration, <CTRL-G> was keyed to activate the Group command, the string <QL> was typed onto the input line, then the up-cursor key was used to set the FIELD number to 00 to ensure that records with <QL> *anywhere* in them were selected. This latter action is important, because Grouping on any field other than 00 only works if the string you enter as the Grouping criterion is at *the start* of the field, and the <QL> string was part-way through FIELD 04 in these records.

Keying ENTER twice causes Grouping to occur, from the start of the database. In this case, the total number of records resulting was about 30, but that was whittled down to 12 using the <CTRL-X> command; some of the people in the 30 might not have wanted their addresses printed here (the editor for instance!). You Xclude by displaying the record that is *not* wanted in the Group or Xclude commands — all you have to do to get all records back is key <CTRL-K> to Kill the Group.

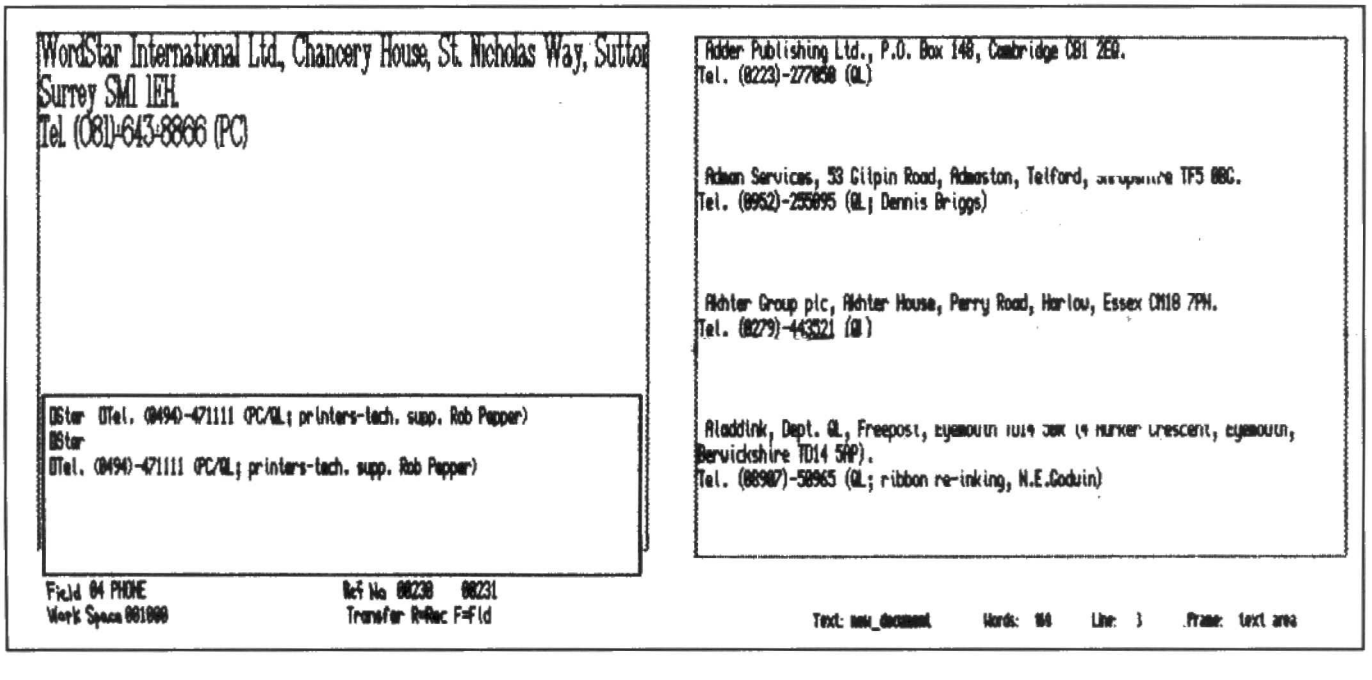
You are now ready to print, as far as the program is concerned, but the problem with printing this way is the format of the text on the page. FlashBack itself (without the Report Generator functions) does not permit you to send instructions to the printer for formatting, so what comes out on a dot-matrix printer is usually 10-pitch (Pica) print, running from one edge of the paper to the other. This is easily altered by sending the appropriate commands to the printer from the SuperBasic command line first. Press ESC (then CTRL-C if necessary) to get out of FB and into SB, then type and ENTER a line such as the following one:

OPEN#7,SER1:PRINT#7,CHR$(27)&CHR$(108)&CHR$(10)&CHR$(27)&CHR$(77):CLOSE#7

This sets the left margin to 10 characters, and the typestyle to 12-pitch Elite. Note that you may have to use different codes if your printer is not "Epson-compatible". You can then go back to FlashBack and print from there, using the new printer settings.

The second illustration shows the printout of the Group, without any further manipulation. There are a couple of points to note on the printout. The first field (and the second in one case) is printed starting one character to the right of the start of the line; this is very likely not sufficiently annoying to be worth trying to deal with. There are three blank lines between each pair of records. This point may be of enough importance to cause the user to try the "transfer method" of printing, via another program, as detailed below.

To transfer the file to text[87], or any other WP/editor program, use <CTRL-P> again but change the device to whatever you wish to use for temporary storage; RAM disk is the fastest. You have to add the file name also; for example, instead of SER1, set the device to RAM_PRINT. All the comments about Grouping apply as before. After the Grouped database has been printed to RAM disk, you can Import this file into T87 using the command <F3 F I A n>, that is, Import it as an ASCII file, in the "normal" (line-by-line) fashion. Illustration three shows the file as it looks after Importing



```
 Adder Publishing Ltd., P.O. Box 148, Cambridge CB1 2EQ.
Tel. (0223)-277050 (QL)


 Adman Services, 53 Bilpin Road, Admaston, Telford, Shropshire TF5 0BG.
Tel. (0952)-255895 (QL; Dennis Briggs)


 Akhter Group plc, Akhter House, Perry Road, Harlow, Essex CM18 7PN.
Tel. (0279)-443521 (QL)


 Aladdink, Dept. QL, Freepost, Eyemouth TD14 5BR (4 Hurker Crescent, Eyemouth,
Berwickshire TD14 5AP).
Tel. (08907)-50965 (QL; ribbon re-inking, N.E.Godwin)


 Alexander, John,     Fieldhouse Drive, Telford, Shropshire TF2
Tel.                (QL)


 Ark Distribution, Corve Farm House, Corve Lane, Chale Green, Isle Of Wight.
Tel. (0983)-79496 (QL; Archivist etc)


 Ashcroft, T,     Mitchell Avenue, Newcastle-upon-Tyne NE2
(QL; scientific references database)


 Aholasoft, Kirjurintie 3, SF-05400 Jokela, Finland.
Tel. ? (QL; Jussi Koskinen, Sam The Little Spaceman)


 Armin Breuer vA. GBR, Postfach 2234, D-6104 Seeheim-Jugenheim 2, Germany.
 Tel. (01049)-6257-7244 (QL/Thor:Armin Breuer, Dirk Schafer, Andreas Brev,
Stephen Michels)


 Atkinson, Joe, 36 Ranelagh Road, Ealing, London W5 5RJ.
Tel. ? (QL; ROMs, mdvs)


 Astracom
Tel. (0480)-412884 (QL; modem)


 ABS Computer Supplies, 4 Shouldham Street, London W1H 5FG.
Tel. (071)-224-8320 (QL; discs, ribbons etc)
```

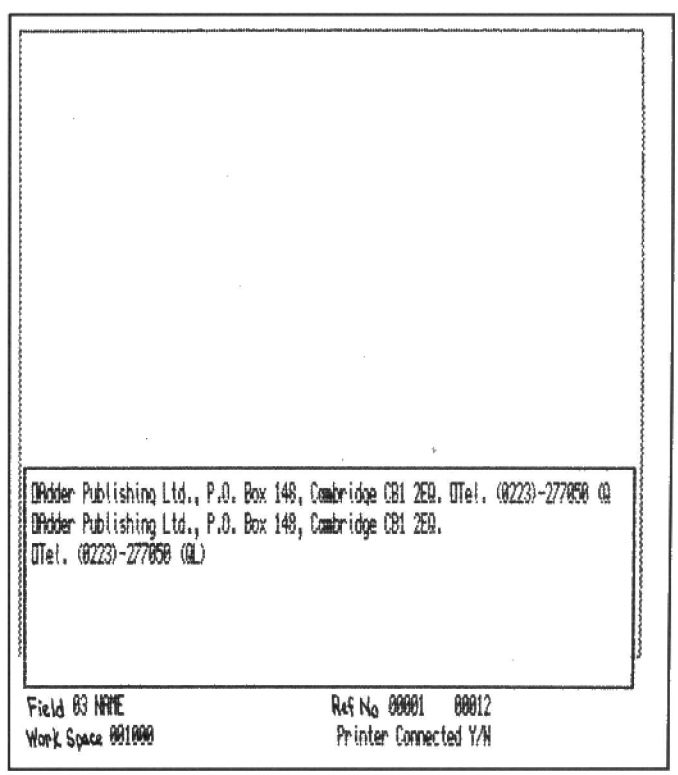
into T87. Bear in mind that the right margin set in T87 may not match that used in FB and you may have to adjust it to get the display right. The three blank lines are still there between records, and you may choose to remove some or all of them; you can also do any other editing of the records that may be necessary, such as pulling the first lines back to the left margin.

Transferring a database this way has the merit of disposing of FB's own formatting codes. You don't need to get rid of unwanted material at start and finish of the file, once it has been Imported into T87. In terms of time and convenience, there is little to choose between the transfer method just outlined, and direct Import of the _DBA file into T87, as described below.

It is desirable to set both the typestyle and the margins you wish to use before reading any file into T87. Use the same command as when reading-in the Printed file: <F3 F I A n>. A point to remember about T87 is that the set allocated memory (using the <F3 C P M nn> command) may have to be considerably greater than the size of files to be Imported. The database used here occupied about 50KB on disk but required a memory setting of over 150KB in T87. Even though you may have grouped the file in FB, you will get the whole of the database when Importing the _DBA file into T87. This may be acceptable, but it can mean a lot of manipulation if you only want a handful of records out of hundreds. You may have to use the Search and Block Copy functions several times to get the records you need all together.

When you read a _DBA file directly into T87, you need to delete a block of information at the start of the file which is for T87's own use and of no interest when printing. Move down the file until you find where the first record of your database is, then go back to the top and use the Block Delete command to get rid of everything prior to that line. Do the same thing at the end of the file, to remove unwanted material from there also. Record fields should be on separate lines, with no blank lines between fields or records, provided you selected the "Normal" option when Importing the file. You are then ready to print, or you may want to edit out the odd spaces on the first lines of records, select particular records, etc.

There is another way of handling printout. You can make use of FB's ability to "drop" records or fields into underlying programs, and transfer records that way, one at a time. The fourth illustration shows one record transferred to T87 this way, with the FB window at the bottom of the screen ready for the next transfer. Where name-and-address records are concerned, it may be desirable to reduce the space taken by the printout. One way of compressing the output is to use condensed (17 characters per inch) print. This can be sent from the SB command line, and you could do two separate runs direct from FB, winding the paper back between them and resetting the left margin so that you get two columns. Alternatively, once the records have been transferred to T87, you can use the Columns function there, together with condensed print. Don't be afraid to experiment, but use a *spare copy* of your treasured database file . . .



DIY TOOLKIT

Simon Goodwin gives QL and Thor drives the DIY treatment.

This article explains how you can accelerate QL and Thor disk access, and check the contents of any drive quickly and comprehensively. CHECK_DISK_BAS suits almost all QL floppy disk systems, while FAST_COMPARE_BAS works with microdrives, ram disks, floppies or even hard disk sub-directories.

The key to speeding up disk access lies in anticipating the hardware and asking for the right thing at the right time. FAST_FORMAT_BAS re-programs the Qdos disk allocation scheme, giving faster access *without* sacrificing compatibility. The new fast format disks are interchangable with old format disks, but load and save large files much more quickly.

These programs are written in Basic, as direct access to QL disks is done the same way from SuperBasic as from assembler. There's little point in writing machine code when compilers can do it for you; a well-designed program will probably spend most of its time waiting for the drive.

QL World has printed many articles about disk expansion, but these have tended to aim at hardware, discussing interfaces, cables and drives. I shall focus on software. The QL harbours some neat design ideas; it's amazing what you can do if you know how.

Bootstrapping

This article is primarily written for those who have a disk system up and running, and want to get the best from it. It helps to define some mechanical terms first. The data on any QL disk is held in concentric rings, or 'tracks', on either or both sides of the disk; the number of tracks on each side is usually 40 or 80.

Every track holds nine 'sectors', each of 512 bytes, giving a top capacity of 80 tracks = 2 sides = 9 sectors = 512 bytes = 737280 bytes, or 720K.

Other information is recorded on the disk when you format it, in-between the sectors, so the system can compensate for small variations in rotational speed and identify any track or sector as it passes the sensing head. The extra data explains why a 720K QL drive is said to have an 'unformatted' capacity of one megabyte.

The QL scheme may seem wasteful, but it has compensations. Commodore's AmigaDOS (developed by Metacomco in Bristol) dispenses with most of the sector gaps, allowing 880K in lumps of 5.5K, one lump per track on each side of the disk. The Amiga can fit 160K extra on a disk, but programs have to read a full track, even if they only want a few bytes.

The larger the lumps, the more disk space can be used by files, and the faster large files can be accessed, but the smaller the lumps, the more files you can get on a disk; lumps are rarely shared, as that would make individual files hard to find or extend.

As a file grows, previously unused sectors must be allocated to it, and the usage of each group of sectors must be recorded. On QL disks, allocation is in groups

of three sectors, so that even an empty file uses three sectors (1.5K).

Files of more than 1472 bytes use six or more sectors; 64 bytes at the start of each file are used for a dummy directory prefix. 64+1472=1536, the allocation size in bytes.

This 'granularity' reduces the number of groups on the disk, and hence cuts the processing overhead involved in reserving or discarding storage. A 1440 sector disk can hold no more than 480 files (1440/3 is 480).

The practical limit is a little lower: when you format a disk to 1440 sectors, up to 1434 are available for files. The other six are used by the map and directory.

The first 'missing' group holds the **disk map** with the disk name and format details, plus three bytes for each of 480 groups, recording the file number and position for each.

The other group is file zero – the **directory,** containing a 64 byte Qdos file header for every file on the disk, including itself. This file occupies one group for every 24 files on the disk. File headers are discussed in *QL World* – February 1988, and Volume F of *DIY Toolkit.*

Most computers cannot use a drive until the user notes its details in a 'system file', like CONFIG.SYS on MSDOS. This is often obscure and inconvenient. The QL is a better judge of a disk than many users, so Qdos works out the number of tracks and sides on a disk or drive automatically.

When you present a new disk the system interrogates the drive to find out its capacity and the rate it can 'step' from one track to another. This explains the rattle and buzz you hear as the disk starts up. This system of trial and error is one of the QL's finest features; it's fast, flexible and gets the best out of the machine and your drives.

The data format on a QL disk was standardised by Sinclair, but the codes and commands were not. More than a dozen QL disk systems have been marketed since 1984; most of these use variants of Tony Tebby's QFLP system rom, but *Quest, Dattel, MCS* and *Sinclair Research* opted for alternatives.

My programs assume you have a QFLP rom, or compatible system, though FAST_COMPARE_BAS will work on any machine with the toolkit command WSTAT. FAST_FORMAT and CHECK_DISK need the QFLP extension that allows you to read and write sectors as 512 byte strings; you're OK if your machine accepts:

OPEN_IN #3, "FLP1_*D2D"

This command gives channel #3 exclusive control over sectors of drive FLP1. The OPEN_IN stops old, incompatible systems creating a file called "FLP1_*D2D" but it does not stop you writing sectors or allow access to other tasks.

Channel #3 can then read and write any sector, given only the sector number (1-9), track (0-79) and side (0 or 1). The mechanism to send this disk address is rather eccentric; you multiply and add the component values, and use the composite thus found to set the file pointer.

SET_POSITION #3, SECTOR + (TRACK * 256 + SIDE) * 256

SET_POSITION is a *Turbo Toolkit* command; *Toolkit 2* users can substitute GET and PUT, prefixing the pointer value with a backslash (\), located above the Enter key on a QL.

At a pinch you could use PAN and SCROLL, but it would take 160 successive integer offsets to get from track 0 to track 79, and that seems gratuitous.

Some QL disk systems have inadequate

documentation, so there's extra information on the DIY Toolkit disk, including standard directory and sector map details. *Quanta* members with an interest in other disk systems can obtain source and compiled code for my TRS-80 *MultiDOS* reader, from the group library.

These programs suit controllers from Miracle Systems, PCML, Computermate, Cumana, Silicon Express, Technology Research, Sandy, CST and Thor International. Suitable upgrades to the QFLP standard are available for Medic systems, and Sinclair's own model, produced by MicroPeripherals. Call Care Electronics for upgrade details.

Old MicroPeripherals roms call the disk device FDK, and include DGET and DPUT commands, which read and write individual sectors, rather than the QFLP equivalents. The French Dattel disk system opts for FLD1 and FLD2, has its own direct sector commands RS and WS, easily adapted.

The expansion unit from Micro Control Systems was designed and programmed by Mark Snape; early versions included many good ideas, but fell short of the QFLP standard. Users tell me it has been fully compatible for the last couple of years, once the MCS Toolkit rom is fitted.

As far as I know, the only QL DOS that refuses to read sectors directly was the first to arrive. Quest's oddball interface was launched, amid a deluge of green plastic leaping frogs, in 1984. This system keeps its operating code on microdrive cartridge, rather than rom, and gets confused by files longer than 32K.

I am keen to hear from anyone using disk systems I have not mentioned, like the QL interfaces advertised by Kempston Micro Electronics.

Like all popular eproms, QFLP software comes in many versions. In general high version numbers are best, but later changes are often intended to compensate for the quirks of particular drives. It may not be worth seeking an upgrade if your drives are already reliable.

Version 1.07 was the first to support direct sector access, and appeared in the original CST Qdisk interface. 1.08 and 1.09 fixed problems with LBYTES from microdrive and access to partly-corrupted disks. Version 1.10 was the first able to read and write old 'single density' disks, on interfaces with an appropriate control chip.

Version 1.11 was functionally identical to 1.10, but the code in the rom was reorganised. Version 1.12 featured much-improved detection of the drive's optimum stepping rate. QFLP 1.13 incorporated tweaks to suit a batch of Mitsubishi drives, encased by many suppliers, which provide unusually short 'index' pulses.

Version 1.14 is a major upgrade. It adds commands to control speed and security, and set up obscure track counts. See the details of FLP_OPT or FLP_TRACK and FLP_SEC in your disk manual, or on the DIY disk. The FORMAT command was re-written and error checking improved.

Troubleshooter recently doubted that anyone might format to 360K in a 720K drive, but I find it useful. If you give QFLP a disk name that has an asterisk as an eleventh character, the disk is formatted on one side, giving half the capacity in about 60% of the time:

FORMAT "FLP1_SingleSide*"

This can format a disk for a few files quickly, or for a single-sided drive.

QFLP 1.14 fixed a silly bug in FORMAT, which occasionally produced a single-sided format by chance. The correction stops QFLP looking for the asterisk unless an eleven character name is supplied.

The toolkit commands WTV and WMON have similar bugs; they read their parameters from memory, even if they are not supplied; this causes inconsistent results which depend on the values lying around in memory.

QFLP V1.14 fixed a serious bug that could destroy the disk directory if you accidentally tried to load a null file (zero bytes) with LBYTES or EXEC. V1.15 of QFLP was adapted for the 1772 interface chip, which replaced the obsolescent 1770. Error-checking is slightly refined, and an extra 6K of small files can be stored. Earlier software would not save small files once 714K was full, because of a bug in the track zero handling.

QFLP 1.16 avoids confusion if you swap disks with files open, or add toolkit commands with a program loaded. 1.17 is the same, apart from the added ramdisk, and 1.18 is more tolerant of dodgy drives. V1.18 to 1.21 are identical, apart from the supplier. I hear that 1.22 to 1.27 have small changes to suit *Minerva*.

Miracle Systems now supply V1.28 with their redesigned Trump Card; this can control up to four floppies. It also incorporates code to control WIN1, Miracle's hard disk drive, which used to load control software from a floppy.

CHECKDISK

PC users have a command CHKDISK which searches a disk for faults. **Listing 1** is CHECKDISK_BAS for the QL. The program attempts to read every sector, and reports unreadable tracks and sides by number.

If CHECKDISK reveals a problem, copy the files onto a fresh disk and start again. Some files may be unreadable because of bad tracks or sectors, but you should be able to retrieve most. Special recovery programs may be needed if directory track zero is damaged.

FORMAT tests the whole disk surface, and locks out parts that do not hold the format pattern. Data can be corrupted later, by software errors, mechanical wear, magnetic or electric fields, running over the disk with your chair, or turning drives on and off with disks inside.

CHECKDISK can re-assure you that your data is still readable, or help to nip problems in the bud – before your backup copy suffers the same fate! It's a good idea to run CHECKDISK before making important backups, or after any event that seems hazardous, in retrospect.

QFLP does not re-read, or 'verify', the sectors it writes; that would make it slow. The only way to be sure a file has been saved correctly is to re-load it. The MCS interface has VER_ON and VER_OFF keywords to control write verification.

CHECKDISK uses WHEN ERROR to trap and report disk errors without stopping the program; so I regret that it does not work on "AH" and "JM" roms. WHEN ERROR can easily crash Qdos if you forget to turn it off and make changes or load another program.

Another bug means the interpreter may lose its place in the program and lock up if a trapped error occurs in an expression. CHECKDISK avoids error-prone expressions, so it should not crash your machine.

The program listed checks 80 tracks and both sides of the disk. Adjust lines 250 and 280 if you are not using a 1440 sector format. For a 40 track single sided drive, substitute:

250 FOR true_track=0 to 39

280 FOR side=0

```
100 REMark QL/Thor CHECK_DISK
110 REMark Copyright 1990 Simon N Goodwin
120 :
130 WHEN ERRor
140   IF ERLIN=300
150     PRINT "Error on track ";true_track;" side ";side<>0
160     bad=bad+1 : CONTINUE 320
170   END IF
180   PRINT "Error at ";ERLIN;:REPORT #1 : STOP
190 END WHEN
200 CLS
210 DIM drive$(8) : bad=0
220 drive$="flp1_" : sector$=FILL$(0,512)
230 OPEN_IN #3,drive$ & "*d2d"
240 CLS : CLS #0 : PRINT #0;"Checking track"
250 FOR true_track=0 TO 79
260   AT #0,0,17 : PRINT #0,true_track;
270   track=true_track*65536
280   FOR side=0,256
290     FOR sector=3,7,2,6,1,5,9,4,8
300       GET #3\sector+side+track,sector$
310     END FOR sector
320   END FOR side
330 END FOR true_track
340 PRINT bad;" fault";
350 IF bad<>1 : PRINT "s";
360 PRINT " on ";drive$
370 CLOSE #3
```

A bug in the current Thor rom, Argos 6.4.1, means that GET will not read sectors into strings that have been dimensioned, so this example uses FILL$ to pack a string to the required size.

FORMAT records sectors on the disk in ascending order, numbered from 1 to 9, but CHECKDISK reads sectors in scrambled order to speed up access. As it takes time for SuperBasic to process each sector once it has been read from the disk, the software must wait for a complete lap of the disk before the next numbered sector comes round again.

```
100 DEFine PROCedure FFORMAT
110 REMark VERSION 3.5 30/8/90 (C) 1989,90 Simon N Goodwin
120 LOCal i,map2,map3,s1$,m1$,m2$,k$(2)
130 CLS #0
140 INPUT #0,"Put an EMPTY formatted disk in FLP1 & press ENTER: ";k$
150 REMark Read directory
160 OPEN #3,"flp1_*d2d"
170 GET #3\1,s1$
180 REMark Extract position of logical sectors 2 & 3
190 map2=CODE(s1$(42))+1
200 map3=CODE(s1$(43))+1
210 GET #3\map2,m1$ : GET#3\map3,m2$
220 REMark Set offset used when stepping between tracks
230 s1$(39)=CHR$(0) : s1$(40)=CHR$(1)
240 REMark Set up LOGICAL -> PHYSICAL translation
250 FOR i=41 TO 49 : s1$(i)=CHR$(i-41) : REMark 0-8
260 FOR i=50 TO 58 : s1$(i)=CHR$(78+i) : REMark 128-136
270 REMark Set up PHYSICAL -> LOGICAL translation
280 FOR i=59 TO 76 : s1$(i)=CHR$(i-59) : REMark 0-17
290 REMark Write new data
300 PUT #3\1,s1$ : PUT #3\2,m1$ : PUT #3\3,m2$
310 CLOSE #3
320 CLS #0
330 PRINT #0;"Fast format ready."
340 END DEFine FFORMAT
```

At best, a SuperBasic loop loading sectors in order from 1 to 9 can only read 2.5K per second, or one sector every 200ms. It may be slower still, after taking account of extra delays stepping between tracks and sides.

CHECKDISK aims to read the whole disk and report bad tracks as quickly as possible, so it checks sectors in scrambled sequence, reading every fourth sector till all nine sectors have been read.

The interleaved sequence used was found by trial and error, and gives quick access on standard QL disks. It can check 720K in two and a quarter minutes, while sequential reading would take six and a half minutes.

FAST FORMAT disks work as normal on QL and Thor, but the speed of programs and operations are different because the interleaving of sectors is changed. The QL is quite flexible; I have used various interleaves on my last four work disks for months without problems, using QFLP 1.16 and 6.4.1.

I use FAST disks for tasks, screens, and other files that I wish to load in one step, but stick with the standard format for *Quill* documents, as they load faster that way and I like to refer to past articles as I write. If you find yourself pushed for time, I recommend you use a 'quick format' disk for incremental saving as you go along, and standard format for finished work. In the former case you SAVE more than you LOAD, and vice versa.

FAST_FORMAT disks are often slower than the originals on interpreted file INPUT and Psion LOAD – but in my experience they can be 30 to 100 per cent faster at LBYTES, EXEC, SBYTES, SEXEC, Psion document SAVE, and even PRINTing long lines from SuperBasic. The speed on task file-handling and COPY depends on the buffer size and number of contending tasks.

At best, the time to save 500K on Thor XVI floppy falls from 1 minute 25 to 45 seconds. A PRINT loop in QL SuperBasic accelerated from 5.3K per second to 8.3 on a 720K disk, or 4.8 to 6.3K per second on a single-sided disk. Screens load at over 16K per second, compared with 12.5K per second normally. The Thor XVI reaches 18.8K per second, saving about three seconds when EXECing *Xchange* from floppy.

If you read a file more than once the QL will often get the data from 'slave blocks' in spare memory, rather than the drive, so swap disks regularly if you want meaningful timings. It's a small advantage, but it's free, and only takes a couple of seconds to convert a freshly formatted disk.

Conversion of slow to fast format is possible, but you need to read each track into memory, re-order the sectors, and write them all back. It is quicker and safer to copy everything from one disk to another.

Sometimes the computer knows from the start which sectors to save or load, and the memory area to be used – EXEC and LBYTES work like that, and *Devpac, Spy* and Chas Dillon's *Editor* load files this way; so do most Page Designers.

In this case QDOS uses a 'scatter load' system, introduced for the ZX Microdrives. Slave Blocks are not used. As sectors come from the drive they are slotted directly into the receiving memory; loading continues until all the slots are full. You can see this clearly if you save a screen on cartridge with

```
SBYTES MDV1_SCREEN, 131072, 32768
```

then clear the display with MODE 4, and re-load with

```
LBYTES MDV1_SCREEN, 131072
```

Try the same thing with an empty disk, and notice the 'shutter' effect as individual sectors load. The sequence used by SBYTES does not match LBYTES on a normal interleave, so loading is uneven in speed and scrambled in sequence.

Now take a freshly formatted disk in FLP1, and type FFORMAT with FAST_FORMAT_BAS loaded. Save and re-load the display on the modified disk. The image should reappear smoothly and consecutively down the screen, at noticeably higher speed. If it pauses or hiccups part-way down, you need to set a new stepping offset.

The first 64 bytes of any Qdos disk contain details of the format and interleave. The first four bytes of sector 1, track 0, side 0 must contain the characters 'QL5A', to identify the QL format. Then come ten bytes for the disk name, and a host of other details.

The QL lets you configure the order in which sectors are used into any disk. FAST_FORMAT adjusts three entries – the stepping offset and the logical to physical and physical to logical translation tables. The logical to physical table contains 18 entries – one for every sector on each side of a track.

The values determine the order in which the software uses physical sectors. Normally QDOS puts every *third* sector in the same group, so a single track contains three groups, comprising sectors (1, 4, 7) then (2, 5, 8) and (3, 6, 9).

Groups on each side of the disk are used alternately, if it's a double-sided disk. The first three groups on track zero use sectors 1, 4 and 7 on side 0, followed by 1, 4 and 7 on the other side of the disk, then 2, 5 and 8 on side 0, and so on.

This suits many sequential files, like Psion data files, and Basic programs that INPUT and PRINT to disk. The gap of two sectors gives the program about 40 ms to process the next 512 bytes of data, before the next sector comes round. If it takes as much as 50 ms, the sector header will be missed and you'll have to wait an extra 190 ms for the next lap of the disk.

In fact the order in which Qdos uses sectors is configurable for every disk, although few people use anything but the standard 1:3 interleave. FAST_FORMAT_BAS alters the interleave on a QDOS disk to 1:1. You could just as easily change it to 1:2, or 1:4, depending on the speed of your program – or keep different formats for different applications. All QFLP and Thor roms are compatible with this feature now, but this is the first time it has been explained.

Normally the table indicates sectors 1, 4, 7 and so on, as explained earlier. Just to add spice it numbers sectors from zero and adds 128 for the other side, so a typical table starts 0, 3, 6, 128, 131 and so on.

FAST_FORMAT re-programs the table to use sectors in strict ascending order from 1 to 9 on side 0, then side 1 similarly.

The Physical to Logical table is the same thing in reverse – 18 bytes indexed by SECTOR+SIDE*9 to give logical sec-

tor numbers between 0 and 17.

The stepping offset compensates for the extra delay associated with stepping from one track to another, so QFLP uses sectors in staggered sequence from one track to the next. The idea is that the next sector should be ready under the head as soon as the disk has stepped from the last sector of the previous track. If the offset is 3, sectors 4, 7 and 1 will form the first block on track 1.

I use a stagger factor of one sector on fast QL disks, and zero on Thor ones, as this gives the best speed for LBYTES and SBYTES. You may find two or three is better, if you have a slow drive – as a rough guide, these usually rattle when others squeak.

There is no 'correct' value, any more than there is a 'correct' interleave. The ideal stepping offset depends as much on your hardware as your software. If your drive steps slowly, replace CHR$(1) in line 230 with CHR$(2) or even CHR$(3). Fast systems may allow CHR$(0).

FAST_COMPARE_BAS only uses one toolkit command – the ubiquitous and arcane WSTAT, which gives 'Wildcard STATistics' about files on a drive. The program quickly compares the files on two drives, by scanning their directory statistics. This means it can miss very small changes, like a few characters altered or transposed, but it picks up most significant errors.

Like most users with hard-earned data on disk I keep multiple copies and aim to save each file to two or three disks after any major change. Problems usually become obvious when I notice that the number of free sectors, shown by DIR or STAT, differs between media when I expect them all to hold the same thing.

Normally this means I have failed to save a file on one of the disks, or forgotten to save an update, so that one file has been extended but the other has not, causing the discrepancy in drive statistics. Sometimes whole groups of files are missing or outdated. Mismatches may stem from known rom bugs – for instance both SAVE and COPY sometimes fail to report errors, leaving incomplete files.

At such times I need a quick way to determine what's gone wrong. There may be 400 or more small files on one floppy disk, and scores or hundreds on a cartridge.

A full DIY Toolkit disk contains over 150 files, ranging in size from a few bytes to 26K. As the kit grew I needed an efficient way to keep my copies up to date and avoid losing tweaks and improvements dreamt up at the end of a hard day's thinking – when it is easy to forget to update one of the backup disks.

I could write a loop to read each sector from each disk, and compare them, but it takes ages to read and compare 1,474,560 bytes on two drives, and shows up lots of irrelevant differences which might be caused by deleted files on one drive, mismatched file name (so DIY19_DOC does not match Diy19_DOC) or differences in the order of files in the directory. What's more, the drives are unavailable to other tasks while open for direct sector access.

I wanted a method of comparing disks that works in a few seconds, ignores trivial differences, and tells me the disk, file name and expected size for all missing files – and the drive and size of *each version,* when files with equivalent names differ in size. I wanted it to work on any QL, with any drives.

FAST_COMPARE_BAS fits the bill, as long as your system has three drives and the WSTAT command, or an equivalent which reports file-names and sizes on alternate lines. You can get by with two drives if you adapt the program to use pipes and QLINK from *DIY Toolkit's* Volume P. Direct the report and temporary files to pipes, or MEM, instead of the TEMP$ drive.

The program starts by reading the directory statistics from drive 1 into a temporary file. It copies the file names and sizes into arrays FNAM$ and FSIZE: the code uses INPUT to read each name, so it is confused if you use names with line-feed characters inside! If you plan to use the program on files which are open, beware of 'in use' reports, or foreign equivalents.

The second lump of code reads through the statistics for drive 2, and tries to match each file against an entry in the arrays. If the name matches, but the size does not, the code reports:

Mismatched size <NAME> <DRIVE1>

<SIZE1> <DRIVE2> <SIZE2>

Files that appear on one drive but not the other prompt:

Excess file <NAME> <DRIVE> <SIZE> bytes

Add wild-cards or sub-directory names to the device names in line 150 if you want to restrict the names of files checked. The result is a report on device TEMP$, in the file "COMPARISON" – you may wish to change the name.

The programs and text of this article are available on 3.5 or 5.25 inch disk, together with further details of QL disks. The price is £7 Sterling, calculated at three quid for the data and four for the service; everyone except the Post Office benefits if you order other volumes, from past columns, at the same time. For a full list, send an SAE to DIY Toolkit, Cwm Gwen Hall, Pencader, Dyfed, Cymru SA39 9HA.

Suggestions are welcome as ever at the *QL World* address, but please allow a while for them to be forwarded; like other contributors, I do not work in the office.

```
100 REMark QL/Thor FAST COMPARE
110 REMark Copyright 1990 Simon N Goodwin
120 REMark Version 1.1 30/8/90
130 REMark Extensions needed : WSTAT
140 :
150 LET d1$='flp1_' : d2$='flp2_' : temp$='ram1_'
160 LET max%=480 : tab1=16 : tab2=53
170 DIM fnam$(max%,36),fsize(max%)
180 GET_STAT d1$
190 count1%=0 : REMark Number of files in device 1
200 REPeat read1
210   IF EOF(#3) : EXIT read1
220   count1%=count1%+1
230   IF count1%>max%
240     PRINT #0;"Too many files on ";d1$;
250     PRINT #0;" maximum ";max% : CLOSE #3: STOP
260   END IF
270   INPUT #3,fnam$(count1%),fsize(count1%)
280 END REPeat read1
290 CLOSE #3 : DELETE temp$ & "temp"
300 :
310 GET_STAT d2$ : count2%=0
320 OPEN_NEW #4,temp$ & 'Comparison'
330 REPeat read2
340   IF EOF(#3) : EXIT read2
350   INPUT #3,file$,size
360   count2%=count2%+1
370   IF NOT ONd1(file$,size) THEN EXCESS d2$
380 END REPeat read2
390 FOR i=1 TO count1%
400   IF fsize(i)>=0
410     file$=fnam$(i): size=fsize(i): EXCESS d1$
420   END IF
430 END FOR i
440 PRINT #4;\count1%;" files found on ";d1$
450 PRINT #4;count2%;" files found on ";d2$
460 CLOSE #3 : DELETE temp$ & 'temp' : CLOSE #4
470 PRINT #0;"See the file"!temp$;"COMPARISON"
480 BEEP 3000,0 : STOP
490 :
500 DEFine PROCedure EXCESS(drive$)
510 PRINT #4;"Excess file" TO tab1;file$;
520 PRINT #4;TO tab2;drive$!size!'bytes'
530 END DEFine EXCESS
540 :
550 DEFine FuNction ONd1(n$,s)
560 LOCal i
570 FOR i=1 TO count1%
580   IF fnam$(i)==n$
590     IF fsize(i)<>s
600       PRINT #4;"Mismatched size" TO tab1;n$;
610       PRINT #4;TO tab2;d1$!fsize(i)!!!d2$!s
620     END IF
630     fsize(i)=-1  : REMark Ignore this later
640     RETurn 1     : REMark Names match
650   END IF
660 END FOR i
670 RETurn 0
680 END DEFine ONd1
690 :
700 DEFine PROCedure GET_STAT(drive$)
710 OPEN_NEW #3,temp$ & 'temp'
720 WSTAT #3,drive$
730 OPEN_IN #3,temp$ & 'temp' : REMark Rewind?
740 END DEFine GET_STAT
```

TROUBLE

A PROBLE

Bryan Davies considers disks and spell checkers

C.G.H. Services report that the 4th issue of *QL Technical Review* was being run off in mid-July and I'll report on it when a copy is received.

The short article on Page 40 in September *QL World* produced a very speedy response from **John Roberts**, who offers an alternative way of adding fields to Archive databases, but only for users having a copy of *The Editor*.

He suggests Exporting the database concerned, then reading it into The Editor and using the commands given below.

Sample database, as it appears when read into The Editor:

```
"firstfield","secondfield$","thirdfield$"
0, "Seconds","Thirds"
1, "Secondday","Thirdday"
2, "Seconds","Thirds"
3, "SecondMon,"ThirdTue"
4, "Seconds","Thirds"
5, "Seconds","Thirds"
6, "Seconds","Thirds"
7, "Seconds","Thirds"
8, "Seconds","Thirds"
9, "Seconds","Thirds"
```

Commands (the > sign indicates exchanges to be made at the ends of the lines and the lines in brackets are explanatory, not part of the command):

```
t ec>>,"fourthfield">
(add another, numeric field)
rp n ec>>,4>
(add '4' to new field, in each record)
t ec>>,"fifthfield$">
(add another, alphanumeric field)
rp n ec >>, "nothing in particular">
(additional quotation marks for text)
```

The final step is not clear from John's letter, and I have not had time to check it; you have to "take out everything on the last line except ? and write the amended file", the ? indicating something missing from the letter. Write out the new file with a _ LIS extension from The Editor, then Import it back into Archive.

The comment in the "Response to Readers' Letters" section about a program ordered from PDQL concerns *HardBack*, written by Chas Dillon for hard disk users. After waiting for well over a month for a copy, one arrived from PDQL the day after an associate complained about its non-arrival (it is his hard disk I am trying to set up). As the program implies, it deals with backing-up files from hard disk, and I hope to report on it at a later date. I had hoped the program would also provide more-general utility functions, to make the hard disk easier to use, but that is not the case. However, the instructions are helpful to anyone trying to understand the workings of the software provided with the Miracle hard disk.

If you are looking for odd items of hardware, try **Adman Services,** which has a large variety of QL (and other) oddments. For example, they have MP disk interfaces, with a choice of rom chips, and dual disk drives plus the MP interface, with box and power supply, for about £150. They can also provide an eprom board, to allow Qdos and other routines to be accessible. The phone number is 0952 255895

Readers' Letters

Differences between printers account for a large proportion of the problems users have. What works well on one printer may initially seem useless on another. A couple of recent examples concern the program *TechniQL* and the optical scanner reviewed by Mike Lloyd a few issues ago. The saga of **J. Roy Goodall's** difficulty printing from TechniQL continues, and it hard to see a way of bringing it to a happy conclusion, because he is so far away (Belize) from the supplier, **T.K. Computerware.**

In view of the fact that the program has been around for almost as long as the QL, it can reasonably be assumed that printout can normally be obtained from it, but Roy cannot get printout to either an Epson LX-800 or a Tandy DMP-130A. T.K. supplied another program copy, and the result was apparently the same, making it seem unlikely the fault lies with the program copy itself. The serial-parallel interface used with the printers has been replaced; hopefully, that eliminates it as a potential source of the problem.

Printout from *Easel* and *Eye-Q* on the LX-800 "presented no problems". The LX is not (so far as I am aware) a fully "Epson-compatible" printer, despite being manufactured by Epson. I have attempted to use an LX printer-driver with an Epson-compatible printer and got decidedly erratic results. Does anyone know of differences between Easel/Eye-Q and TechniQL, in the way they treat printers as being Epson-compatible? One other thought that comes to mind at present is the likelihood that Roy's QL has a JSU rom (the type fitted to QLs sold in the USA); it is known that screen graphics can look quite different with the JSU, compared to the JS UK rom, and it may be that the output to the printer is similarly different. In addition, it is thought that some US-specifications differ from UK-spec ones. Perhaps US readers have some thoughts on printing from TechniQL?

The scanner is a printer-dependent device in so far as the software has to drive the print head. To be able to do this, the software has to be supplied with various movement codes specific to the printer being used. In the case of **E. May,** the printer is a Seikosha GP-100A Mark II, and he has not succeeded in getting the print head to scan his documents. The head moves a little, then stops. The printer details have been sent to the manufacturers of the scanner, and comment from them is awaited. In general, the scanner does a good job; Mike Lloyd was impressed with the one he reviewed, and I am informed that it can do a good job of displaying a £5 note in colour!

Keyboard Products have refunded **Richard Alexander's** money, having been unable to supply the ordered keyboard in time (but he says he may re-order). As reported last month, they stated they have completed the redesign which caused delay, and should be delivering PS/2 keyboards in a reasonable time now. As of late August, there has been no further comment from **Michael Jackson** regarding his order for this keyboard, but there have been no other complaints.

R.W.S. Dand says he contacted his credit card company about the double charge for goods supplied by **PDQL** and the second debit has been refunded, but the credit card company said they had not been repaid by PDQL. **J.S. Hay** now has a working Rename routine, supplied to him by **Brian Hedge**. Comparison of their typed-in lines revealed an error in one data value. There was more to it than this, however, as Hedge had written a 3-line routine to get around the problem many earlier QLs had with the CALL command not working properly, and this was necessary to enable the Rename routine to run on Hay's QL. **E.E. Stocker** has received a refund and disk from **Sector Software**, following a mix-up about orders he placed with them. **J. Roberts** has contacted **Psion**

about corruption on his Psion Chess cartridge, and they have offered to help. Although Psion do not now bother with the QL market, they have been helpful in dealing with queries on various occasions, and they don't seem to have "washed their hands" of the QL.

A problem of a different nature, to which I know no answer, was put by **C.G.H. Services**. In response to my repeated advice to pay for goods by credit card, they state that small businesses are not considered suitable to be given a credit card "franchise", so that I am effectively advising readers not to buy from small businesses. My understanding is that card companies do not refuse facilities to small businesses but, understandably, they may refuse in cases where they feel a business is not sufficiently long-established or cannot provide suitable references. I did not mean that buyers should make use of a credit card a hard and fast rule (especially now card companies are starting to charge for the card, and its value has dropped correspondingly). If you spend, say, £5 to £20 as you might with several small suppliers most buyers can afford to take the chance of losing that amount, and pay by cheque or postal order, etc. It is when you order expensive goods that the use of a credit card becomes highly desirable. It is for the individual to decide at what level loss of the money would be painful. Credit card companies may set a limit below which they do not give refunds, and that figure has typically been £100, although that does not seem to mean refunds of lower amounts will automatically be refused.

It would certainly be untrue to imply that small companies are inherently less trustworthy than large ones. In any event, all the suppliers we are familiar with on the QL scene are small, by normal business standards. Large adverts and familiar names are not guarantees that your order will be despatched but, if the new owners of *QL World* (the Maxwell group) ensure that advertisers pay promptly for their adverts, there will much less chance of adverts appearing in future from suppliers in the process of going out of business.

The PC supplier which presented the most unbelievable face to me, resulting in my having to claim a refund from a credit card company, now provides a large advertising spread in the PC magazine I read regularly, and similar adverts in several others.

One has to "test the water" at some point, by sending money and hoping for something in return. Preferably, one should have favourable comments from other buyers to give confidence. Naturally, those of us who write for QL World come into contact with suppliers,and form our own opinions of those who seem trustworthy; inevitably, we deal more often with the bigger suppliers, because they have more hardware/software to be reviewed, and because more readers write in about them, since more readers buy from them. At this point in time, the main suppliers all seem both trustworthy and commercially sensible. That is, they normally supply what is ordered, only cashing cheques when they feel certain the goods are available for despatch, and they keep their finances in sufficiently good order to be able to meet their commitments. Several suppliers have reported a downturn in orders over the past few months, and an increased level of "suspicion" on the part of buyers. That is, because of the way two suppliers treated their customers last year, and early this year, good suppliers are now suffering loss of business, and finding that buyers are tending more to want to collect goods personally rather than order them by mail.

What can one say, in hindsight, about PDQL and SUB? As a supplier, PDQL did not always seem as "professional" as some others, but it never gave us the impression of being unsafe to deal with, not until about the time the letters of complaint started arriving at the magazine. The situation is still quite unclear, as an associate of mine recently (late August) called PDQL to complain about ordered goods not being supplied, and was assured one particular program would be sent that day and it was duly received the next day.

SUB didn't seem "right" to me, from the start, but you can't advise readers to steer clear of a supplier simply because your own intuition has doubts. The concept of an answer service for users with problems was laudable, but the chances of success with such an enterprise struck me as small. Eidersoft tried it earlier on, and came unstuck. Taking money on the promise of supplying help is a shaky premise, for the supplier as well as for the punter. As the supplier, if you subsequently find that you don't have the time or the expertise to provide the answers the subscribers need, you soon end up having spent the money they sent you, and with a mass of complaints and calls to deal with. Once you let correspondence build up to a significant level, you won't get any work done. A supplier that gets even one letter to deal with for every 50 to 100 orders despatched may find it hard to keep order processing efficient, and to make money and this is also a reason why it can be difficult to get replies to letters even from "good" suppliers.

Insurance is a subject we all tend to fight shy of, sometimes to our cost. Recent correspondence concerning an expensive item returned to a supplier as defective, and received by the supplier in a damaged condition, highlights this aspect of mail order business. If the goods are expensive, insure them. Don't expect the supplier to have insurance cover for goods that might be returned to them! Make sure the insurance is for the full value of the goods, not for some nominal amount which may be far less. It is also essential to advise the supplier of what you are doing, and to give the supplier first look at the goods which are thought to be defective. If you feel it is necessary to bring a third party into the matter, choose someone who is professionally-qualified to give a full written report on the claimed defect, and who will submit a proper report that would be acceptable to a Court as "competent, *independent* professional opinion".

A supplier called to advise that we might get a complaint from a QL user who had ordered certain programs, changed his mind and cancelled the order on the basis of "expert opinion" on the programs concerned, and then changed his mind again and reinstated the original order. The supplier might not now be keen on dealing with the order at all, as the amount of time spent so far, and the distinct possibility of further queries, suggests the sale would hardly be a profitable one. A few words on the need this particular user apparently has might answer questions for others. The stated requirement was to be able to transfer text files between QL and PC, in both directions, maintaining typestyle enhancements (eg italics, bold). The program to be used on the QL was text[87], and WordStar and/or WordPerfect seemed to be the other WP program to be used on the PC.

The first point is that direct transfer of files, without any attendant "translation routine" being used, would enable text to be transferred with little difficulty, but enhancements would be lost in the process. Whatever you put into the original file, the version of it received by the target pro-

gram would be devoid of the correct codes that tell the printer when to change size, print in bold, use an italic character set, etc. If the user accepts this limitation, and is prepared to reinsert printer codes using the target program, the transfer process is quite easily accomplished using *XOver*, *MS-QLink*, or similar programs.

The transfer process does not actually remove all non-text codes; if you want files transferred from the QL to be as "clean" as possible when they are passed to a PC, the obvious program to use is The Editor. It keeps its files in a simple Ascii-text format and doesn't insert lots of formatting codes for its own purposes. *text*[87] does use its own formatting codes; if you transfer a standard _T[87] file to a PC you get rather a mess, which might take you a very a very long time to knock into acceptable shape. You should use the Export-Ascii command in text[87] to make the file to be transferred as clean as possible before using XOver or MS-QLink — to transfer it. Using the Export command does not, however, get over all problems associated with print codes. As an example, the dash (hyphen) character used in text[87] is different from that used in WordPerfect 5.0; although the dashes remain in the transferred file, you can't use the Search command in *WordPerfect* to find them, because it simply does not recognise the code used by text[87] (this is an oddity of WordPerfect, not a fault in text[87]).

When *WordStar* files are transferred from PC to QL, the last character of each word gets altered.

You can deal with some of these problems by writing a macro instruction or command file to make necessary conversions once file transfer has been completed, but this is clearly only practicable when the target program is capable of recognizing the codes which need changing. For example, when a WordPerfect file is saved in Ascii form and transferred to the Editor, the only tidying-up work needed may be to remove end-of-line codes (displayed as "splodges" by The Editor), and The Editor is well-suited to doing this job with the minimum of activity on the user's part. Most WP programs (including text[87]) are not designed to do the manipulations that The Editor is capable of and they don't recognise many of the codes that it can use.

Where print codes in a text file are explicit — that is, you can see the codes on the screen display — and you can choose your own codes, you are in a position to select codes which the target program will recognise. Again, The Editor is the obvious program, because any typestyle enhancements you insert into Editor files have to be identified by on-screen codes, which you can select to suit your purpose. Quill uses embedded codes for some typestyles (underline, superscript, subscript and bold), so you will have trouble with these, but you can use the Translate entries (when creating the printer-driver) for other typestyles and choose codes to suit.

Text[87] is the best QL WP program, from the point of view of handling many typestyles, but the codes are all embedded. My experience importing text[87] files into WordPerfect is that — whichever method is used — tabs and typestyle codes are lost and, with one method of importing, "real characters" (such as the degree sign °) may also be lost; the end result is satisfactory for my purposes, but there can be a fair amount of time involved in putting enhancements back in. Much the same applies with Quill.

Complicated as it may sound, there is merit in using The Editor as the transfer agent, but using text[87] as your normal document-preparation program. Make the transfers (both ways) via The Editor, to keep the files clean and enable conversions to be made, but use text[87] to prepare text for printing. You can easily Import files from The Editor into text[87].

SUPER BASIC

Mike Lloyd battles with alien forces to develop a playable arcade game written entirely in SuperBasic.

The SuperBasic series has tackled many subjects over the years but it has up to now avoided the specialist subject of games. This has largely come about by accident, but this serious omission is about to be put right with a short series of articles devoted to games written entirely in SuperBasic.

There are four major categories of computer games: the category most people immediately think of when considering computer games is the arcade variety. Arcade games can be further broken down into sub-specialities such as "shoot-em-up" involving the traditional alien hordes, "duck-and-weave" such as Pacman, and "search-the-castle" hybrids which mix arcade style gameplay with elements more commonly found in text-based adventures. The main features of such Games are speed, noise, fancy graphics and a sense of power and danger.

Basic games

In the early days of the home computer revolution arcade games were commercially marketed in simple Basic, but nowadays all professionally-produced arcade games are written in high-speed machine code to provide as much visual impact and realistic action as the cpu will allow. The Basic language is normally thought of as being too slow and too limited for fast games. This is largely true even if the Basic program is compiled, but there has nevertheless been a steady stream of playable arcade-style games written completely or mainly in Basic and published in magazines like *Sinclair QL World*. Just glance at the Microdrive Exchange for examples.

Most Basic arcade games are based on or related to the early efforts of the computer game pioneers because those early games were carefully designed to make the most of the powers of contemporary computers without exposing too many of their weaknesses.

The earliest games played on computers were entirely text-based and they form a category of their own: adventures. Adventure games tend to be of the "dungeons and dragons" genre with a "quest" scenario, but successful adventures have been based upon murder mysteries, spy books and even the highway code. A typical format involves the description of a location and input covering movement, picking things up and avoiding dangers. Nowadays the techniques developed over the years for adventure games are finding their way into computer training packages, with the computer responding appropriately to the trainee's input.

The satisfaction in both playing and writing adventure games comes from the rational, logical representation of a realistic world in which every conceivable option has been catered for by the program. Recent adventure games rely on graphics to convey atmosphere and information, and some have replaced text input with on-screen menus or arcade-style responses. Adventure games become very unsatisfying if a perfectly reasonable action is not permitted because the game designer did not think of it or if the game depends upon meaningless conditions being met – the door will not open unless you are holding a box of matches, for instance.

A third category of computerised game comprises intellectual games such as chess, backgammon, *Scrabble* and even *Monopoly*. The essence of these games is to recreate on the computer screen games which are already familiar to users. The computer can then become a highly capable opponent, using sophisticated "look-ahead" analytical techniques to replace human intellect. Some of these methods have been explained in the series on a draughts program published in previous editions of QL World.

The final category includes all simulators, be they flight simulators, car racing simulators or games which would not normally be considered as simulations, such as computer golf and computer snooker. Quality wargames packages fit into this area of the games market, depending upon whether realism or combat machismo is the aim.

The satisfaction obtained from simulators tends to be in direct proportion to the illusion of reality created by the programmer. There is a good argument that simulations have become too complex and fail to acknowledge that facing a computer screen and keyboard provides much less information than would be available in real life. The weight of a golf stroke cannot properly be gauged, nor can an aircraft's height above the runway be judged, when the only information available is a low-resolution graphic of a small dial.

This weakness of simulations is being tackled on two fronts: improving the displays and changing the way user responses are handled. The Japanese have been experimenting with special seats with screens all around them and an "active" suspension to reproduce cockpits for car, aircraft and spaceship simulators. One of the more promising options becoming available is a set of jumping pads which plug into a computer and detect the user's foot movements. This turns athletic simulations into something much more credible. It cannot be long before even the humble exercise bicycle has a display screen so that users can race against computerised opponents.

Each class of computer game holds its own peculiar challenges, and those of the arcade games demand programming techniques of far greater complexity than those normally expounded in this series. Nevertheless, the first game to be tackled in the Super basic series is very much of the traditional arcade variety, albeit with some interestingly original ideas included within it. Before getting down to detail in the next article it is worthwhile to consider some of the general problems which programmers of arcade games have to overcome.

Top speed

Speed is by far and away the most important consideration when planning an arcade game in Basic. There is a constant conflict between adding special effects and keeping the program short and simple enough to ensure that it is fast enough. The fractions of a second differ-

ence between the speed of an IF command, and its SELECT ON equivalents might not seem very important, but if a comparison is made within a loop at the heart of the program executed many times a second then it might well make the difference between a playable game and a non-event.

Almost everything in a game program's design is subordinate to speed, including conventional programming style. Forget long variable names, neat IF..THEN statements, clear algorithms and the division of a program into coherent procedures. For a Basic arcade game to work the code must be pared down to its most efficient, and every trick in the book to achieve this aim is valid, no matter how unacceptable it would be in a standard business program.

At the heart of a typical game is a loop in which the screen is updated and user input is detected and acted upon. This loop, the main game loop, is where speed counts the most. The main game loop usually breaks down into a body of statements which are executed every time the

Game loop

loop is cycled and a number of statements which are only executed if a condition is met (for instance if the user presses the "fire" button, or an alien hits the bottom of the screen).

The efficiency of the main game loop is compromised by every GOSUB, GOTO, procedure call or user-defined function. Jumps out of the main game loop, by a GOSUB for example, should be ruthlessly excluded from the fixed part of the loop and only tolerated when speed is not vital in conditional parts of the loop. There is therefore not much call for sophisticated structures such as user-defined procedures in an arcade game program except for ancillary features such as displaying a high-score table or animating the dying moments of the user's battlestar at the end of a session.

To keep a Basic game moving at an acceptable pace the number of objects having the appearance of moving simultaneously must be reduced to an absolute minimum, which might mean only two or three items. The Space Invaders format is not really suitable for a Basic conversion because it involves far too many independent aliens (and the products of their firepower) which must each be moved during one cycle of the main game loop. The Pacman format is much more acceptable, with only the one "munchy" controlled by the user and perhaps two or three "ghosties" haunting the maze.

One way of giving the impression that there are a larger number of entities moving on the screen is to create a string with a number of items a few spaces apart. By printing the whole string, perhaps six things can be moved in the time it takes to move one, but of course their relative positions will be unchanged.

Movement does not have to be pixel-by-pixel in order to convince the player that something is actually moving when in fact it is being rubbed out at one location and re-drawn at another. Generally, movement in hops of one character position or one line is acceptable. The time between an object disappearing at one location and being replaced at another should be as short as possible and the time it then remains on the screen should be as long as possible before it is necessary to move it again. The main game loop should therefore be designed so that an item is drawn on the screen, its next location is calculated, and all the other actions which must take place are completed before it is overwritten pending its re-drawing somewhere else.

A potentially lengthy part of any game loop is the need to read keyboard input and respond to it. Generally, this means that the keyboard has to be read for input and, if detected, the input must be tested by a series of conditional statements to determine what action needs to be taken. Where conditions are mutually exclusive, for example where only a left cursor or right cursor keypress could have occurred, the most likely condition should be tested for first and the other possible conditions should be coded in an ELSE alternative, such as:

```
IF fire_button THEN ...
  ELSE IF left_move THEN ...
   ELSE IF right_move THEN ...
   ENDIF
  ENDIF
ENDIF
```

Do not forget, of course, that for some games the most common condition might be no keypress at all, in which case this will have to be tested for before any particular keypress is considered.

Tuning clauses

Where simultaneous keypresses are permitted, which must be true for the majority of arcade games, a different approach must be taken. Each possibility must be tested for in each cycle of the main game loop, leading inevitably to a series of IF...THEN clauses. Even here, however, it is possible to save time by paying attention to the phrasing of the clauses. For example:

```
IF NOT X THEN ...
```

might well be slightly slower than its equivalent:

```
IF X = 0 THEN ...
```

Where ranges are considered the scope for tuning commands to obtain maximum efficiency grows considerably. Ranges might be inclusive, like:

```
IF X>= 7 OR X <= 11 THEN ...
```

or they might be exclusive, like:

```
IF X > 6 OR X < 12 THEN ..
```

There is usually merit in avoiding OR and writing IF statements to include AND instead – it is surprising how often this is possible. Ranges might also be broken down into two separate IF statements, such as:

```
If F X > 6 THEN
  IF X < 12 THEN
  ...
  ENDIF
ENDIF
```

With the QL, of course, it is also worth considering using the SELect construct instead of IF..THEN. Tips as to the faster of the two conditional constructs have been published in a recent SuperBasic article.

Knowing it

In order to extract maximum execution speed from a program and to include as much screen action as possible it is essential that you know your programming language and the capabilities of your computer inside out. The QL has some unusual quirks which generally make it a less suitable platform for computer games. It has no means of detecting whether a character has been printed at a particular location on the screen, for example. However, it has superb conditional structures which permit boolean logic (more of which next month) to quickly act upon keypresses detected with the very fast and capable KEYROW command. It also has an unusual option to the OVER command which gives an additional form of overprinting.

Programmers also need to be aware that procedure and function calls are slower than GOSUBs, that jump to a distant part of a program take longer than a jump to a nearby part of the program, that normal QLs actually perform integer arithmetic more slowly than floating point arithmetic, that vertical and horizontal lines are much more quickly drawn using the BLOCK command than by the LINE command, and so on.

Increasingly these days, knowledge of standard SuperBasic needs to be supplemented with knowledge of whatever toolkits, replacement Roms and compilers are pressed into service alongside it. For example, both the *Minerva rom* and all the Basic compilers perform integer arithmetic much more quickly than floating point arithmetic. Another worthwhile example of useful knowledge is that the *Lightning* screen accelaration program works at its fastest if characters are an exact number of bytes wide (eg CSIZE 0,1) and they are printed at a screen location whose horizontal offset from the edge of the screen is exactly divisible by four.

In next month's article the process of designing an arcade game will be illustrated by a relatively fast and original arcade game programmed totally in SuperBasic.

Programming IN

In the second article on his new C series, Andy Wright moves onto facilities, including variables and control statements.

The previous article on C programming was mainly concerned with the basic structure of a program. We also took a look at how to compile and run a program using the Digital C compiler.

This month we'll take a more detailed look at some C facilities.

The main areas covered are:

a) defining symbolic constants
b) defining and using variables
c) sequence control statements (while and for)
d) conditional statements

Two C program listings are included to illustrate these facilities. The program in **fig. 1** performs a fairly simple task. It counts the number of characters that you type in. When you press the (ESC) key it tells you how many keys you pressed and then stops. If you type in more than 32765 it stops automatically without you having to press (ESC). The program displays each character as you type it – but it is not especially fancy. Many characters on the QL keyboard do not produce displayable characters – these will appear as blobs on your screen. Examples of such keys are the (TAB) key and the arrow keys that usually move the cursor around. Inside our C program the arrows and (TAB) simply get displayed as blobs!

The first two lines of the listing in fig. 1 define things known as symbolic constants. In our program these are named MAZ__INT and ESC. MAX__INT is given the value 32767 and ESC is given the value 27. When the program is compiled the C compiler (parser) replaces every occurrence of MAX__INT by 32767 and every occurrence of ESC with 27. The only exception to this rule is that if MAX__INT or ESC are enclosed within double quotation marks (eg "MAQX-__INT") they are left alone and not replaced).

Symbolic constants are defined using the following syntax:

```
#define SYMBOLIC__NAME value
```

It is conventional to use upper case letters in symbolic constant names. Underscore characters, alphabetic and numeric characters can also be used in SYMBOLIC __NAMES.

The value assigned to the SYMBOLIC __NAME can be almost anything. Usually every #define appears at the start of the program before the main function.

Using symbolic constants can sometimes make programs easier to understand and easier to change. We are using MAX__INT to represent the maximum value that *Digital C* allows an integer variable to have (32676). ESC represents the Ascii character that is produced when you press the (ESC) key on a QL keyboard (Ascii code 27 decimal).

All program variables must be declared before they are used. Usually they are declared at the beginning of the main function – just after the curly bracket. There are a few other places where you can declare variables, but for now stick to the beginning of main.

Fig. 1 declares two variables. The first is named num__characters and is an integer. The second is named inp__char and is a character.

The basic syntax used for declaring variables is:

```
data__typ variable__name;
```

You can declare many variables of the same data type using:

```
data__type variable1, variable2, variable3;
```

When variables are declared in C they are not normally given any initial default value. Unless you assign a value to a variable (see below for details) it will contain an unpredictable garbage value.

Digital C supports two basic data__types — int and char. In future articles we'll take a look at arrays and pointers. Full implementations of C include floating point (decimal) numbers. Digital C does provide some facilities for handling floating point variables – again we'll consider them later.

In Digital C an int can hold a value in the range –32768 through +32767. Some implementations of C can cope with larger values (sometimes they use a data__type known as long int to do this).

A char can hold a single Ascii character. Ascii characters have codes in the range 0 through 255. The QL Concepts manual that is provided with the QL gives details of which code represents each of the characters.

To assign a value to a variable use:

```
variable__name = expression;
```

An expression often consists of constants and/or variables__names linked together by arithmetic operators such as +, −, * and /. Some examples are:

```
/*Assuming x, y and z have been
declared as type int */
x = 1;
y = x + 7;
z = (x + y) * 15;
```

In fig. 1 both num__characters and inp__char are assigned an initial value of zero. One curious point about char variables is how they are assigned values. The statement:

```
inp__char = 0;
```

stores the character whose Ascii code is zero (NULL) in inp__char.

In a similar fashion:

```
inp__char = 65;
```

stores an upper case A in inp__char (Ascii code 65). For characters that can be typed in from a keyboard, however, you can use:

```
inp__char = 'A';
```

This stores the Ascii character A in inp__char. The C compiler can only manage this if you include the apostrophes (single quotes) on either side of the character – if you don't then it expects you to use a numeric value in the range 0 through 255.

The next few lines of fig. 1 print some messages. Last month we covered the basics of the printf function. In fact, printf

```
printf ("words to print and print__formats", expressions and variables
        separated by commas);
```

```
#define MAX_INT 32767
#define ESC 27
/* Defines some symbolic constants */
main ( )
{
    int   num_characters;
    char  inp_char;
    /* Define an integer and a character variable */

    num_characters = 0;
    /* We haven't read any characters yet */

    printf("This program counts the number of keys that you press\n");
    printf("     until you press ESC or type %d characters\n\n",
           MAX_INT - 1 );

    inp_char = 0;
    /* Makes sure that inp_char isn't already ESC by coincidence */

    while ( inp_char != ESC && num_characters < (MAX_INT - 1) )
    /* Loop round until all characters have been entered */
    {
         inp_char = getchar();
         /* Get a character from stdin */
         if ( inp_char != ESC )
         {
              num_characters = num_characters + 1;
              /* Only increment num_characters if inp_char != ESC */
              putchar(inp_char);
              /* Write the character to stdout */
         }
    }
    printf("\n\nThe number of characters entered was: %d\n\n",
           num_characters);
    /* Write the results to stdout */
    printf("Press a key to quit");
    inp_char = getchar();
    /* Wait for key before terminating */
}

/*            Fig 1: Character counting program                */
```

allows you to print a character string and/or the values of program variables. The basic format is on the previous page.

The double quotes must be included. The words to print can include almost anything – including special characters such as \n to force a line feed. You can alsoinclude details to tell printf how to print program variables and expressions. These print-__formats always start with a % sign. In fig. 1 %d has been used. The %d indicates that the variable should be printed as an integer. %c would have told printf to print it as a character. If you want to print a real % symbol using printf you should use \% in the words to print string. There should be one expression or variable for every % format specification.

The printf function always writes its output to a device known as stdout. This is normally the computer screen.

Note that in the second printf the value of MAX__INT –1 is printed. The compiler replaces MAX__INT by its value (32767) and so the coding works. If MAX__INT were within the quoted string of printf the compiler would not have replaced it!

The next part of the program in fig. 1 uses a while loop. The loop in the program keeps going round and round until you press the (ESC) key. The loop used in the program could have been written more elegantly - but we can afford to be more elegant later. The while loop structure is:

```
while ( condition )
{
    statement;
    statement;
}
```

The statements enclosed between the curly brackets are executed over and over

Programming IN C

again so long as the condition is true. When the condition becomes false the loop is terminated and control passes to the statement after the closing curly bracket. The statements within the curly brackets make up what is known as a compound statement.

In a similar fashion to SuperBasic, in C conditions are actually expressions. If the expression evaluates to zero then the condition is false, otherwise it is true.

Some of the comparison symbols in C are different than those used in SuperBasic. For example:

```
!=   is not equal to
==   is equal to
>    is greater than
<    is less than
!    not
```

Using == instead of = when comparing values is something that most newcomers to C find very confusing. Only use = in value assignment statements – if you use it when comparing values you will get unexpected results.

Several conditions can be linked together (as in SuperBasic) using the following logical operators:

```
&&   means AND
||   means OR
```

(two of SHIFT and the key at the top right hand corner of a QL keyboard).

Parentheses () can be used inside complicated conditions (as in SuperBasic). The While loop in fig. 1 keeps going until inp__char is equal to the (ESC) key or num__characters is euqal to the highest valid integer value minus 1. If num__characters were allowed to get bigger than 32767 then it would end up holding an incorrect value.

Within the while loop, the getchar function is used to read a character from the standard input device (stdin) into the variable named num__char. The basic syntax is:

```
variable __name = getchar ();
```

The variable can be either an int or a char. The stdin device is, by default, the keyboard.

If the value of inp__char is not equal to the Ascii code for the (ESC) character the value of the num__characters is increased by one and the character is displayed on the screen using putchar. The basic syntax of the if statement is:

```
if ( condition )
{
     statement;
     statement;
}
```

If the condition evaluates to true then the statements enclosed within the curly brackets are executed. As with the while loop, the condition is actually an expression – if it evaluates to zero then the condition is false, otherwise it is true. You can also use:

```
if ( condition )
{
     statement;
     statement;
}
else
{
     statement;
     statement;
}
```

It is possible to nest if statements within each other.

The putchar function is used to write a single character to the standard output devide (stdout). This means that we get to see the characters as we type them in – the getchar function does not display anything on the screen. The basic format is:

```
putchar (character);
```

The character used in putchar can be:

```
the name of a char variable
an integer representing an Ascii character code
a character enclosed in single quotes
```

Instead of using putchar we could have used:

```
printf ("%c", inp__char);
/* %c prints it as a character – %d would print its decimal Ascii code*/
```

When the while loop is terminated the results are printed together with the message "Press a key to quit". The program then waits for a key to be pressed by using the getchar function. This technique of waiting for the user to press a key before reaching the end of a program is useful if you plan to run the finished product in a windows environment such as that provided with Qram. In a windows environment the program runs within a window and all of the output to stdout appears in the window.

When the program finishes the window is closed and its contents disappear. If you're not careful then the window disappears before you get a chance to look at the results. The extra getchar at the end of fig 1 stops the program from finishing before user reads the final message.

Exercise 1

Try some of the following – it's probably best to handle each of them as a separate exercise and undo the changes before carrying on with the next one.

1. Miss out the brackets from the condition in the while or if statement.
2. Remove the line that sets inp__characters to an initial value of zero.
3. Remove the line that sets inp__char to an initial value of zero. Try running the compiled program a few times.
4. Change the program so that it stops when the user presses 'Q'.
5. Remove the %d print__format definition from one of the printf statements. Also try including more print__formats than there are values to be printed.
6. Write a program to find out what happens when an int variable gets bigger than 32767. Start off by giving the int a value of, say, 32760 and go round a while loop which prints the integer value and then adds one onto it until the user presses the (ESC) key. You needn't bother printing the character that the user types in each time you go round the loop.
7. You can do arithmetic on char variables in the same way as with int variables. Write a program to find out what happens when a char variable gets bigger than 255. Take the same kind of approach as you did with the int example above – but starting from a value of, say, 250. Try to print the value of the char variable as an integer value (%d) and as a character (%c) using the printf function.

The listing in **fig 2** counts the number of characters, words, alphabetic characters, spaces and numeric digits that you type in. You should be able to understand most of it already. It introduces a few new C standard library functions and the structure of a for loop. In the next few paragraphs we will only consider the new concepts.

In fig 2 the main processing loop is driven by a for statement. The basic syntax of this is:

```
for (initial statements separated by commas; condition;
     statements for end of each repetition separated by commas)
{
     statement;
     statement;
}
```

```
#define MAX_INT 32767
#define ESC 27
#define TRUE 1
#define FALSE 0
/* Defines some symbolic constants */
main ( )
{
    int   num_characters,     /* Counts total characters */
          num_spaces,         /* Counts spaces */
          num_alphas,         /* Counts alphabetic characters */
          num_numerics,       /* Counts number of numeric digits */
          num_words,          /* Counts words */
          within_a_word;
    char  inp_char;

    printf("This program counts characters and words etc\n");
    printf("     until you press ESC\n\n");

    inp_char = 0;

    for ( num_characters = 0, num_alphas = 0, num_spaces = 0,
                num_numerics = num_words = 0,
                within_a_word = FALSE;
          (inp_char = getchar()) != ESC;
          ++num_characters )
    {
         putchar(inp_char);
         if ( inp_char == ' ' )
              ++num_spaces;
         if ( isalpha(inp_char) )
              ++num_alphas;
         if ( isdigit(inp_char) )
              ++num_numerics;
         if ( inp_char == ' ' || inp_char == '\t' ||
              inp_char == '\n' )
         {
              within_a_word = FALSE;
         }
         else
         {    if ( !within_a_word )
              {
                   if ( isalpha(inp_char) )
                        ++num_words;
                   within_a_word = TRUE;
              }
         }
    }
    printf("\n\nThe numbers entered were\n\n");
    /* Write the results to stdout */
    printf("Alphas Numerics Spaces Words\n");
    printf("%6d%9d%7d%6d\n", num_alphas, num_numerics,
                             num_spaces, num_words);
    printf("\n\nPress a key to quit");
    inp_char = getchar();
    /* Wait for key before terminating */
}

/*            Fig 2: Multi-counting program              */
```

Programming IN

All of the statements between the curly brackets are executed on each pass round the loop

The initial statements are all executed once before the first pass through the loop of curly bracketed statements. The initial statements are not executed again as the program loops round the bracketed statements. Note that the initial statements are separated by commas! The coding in fig 2 illustrates that you are allowed to use:

```
variable = variable = variable
= expression;
```

to assign values to many variables in a single statement.

The condition is similar to the type used in while and if statements. The loop continues so long as the condition is true. In fig 2 the condition is a little more complicated than the ones that we've seen before. It is:

```
(inp_char = getchar()) !=ESC;
```

The positioning of the brackets is important. You may find that you need to read through the next few paragraphs more than once (slowly) to make sense of them.

The getchar() function requires an empty set of brackets. The outer set of brackets force the value returned by get_char() to be passed into inp_char. The results of this is then compared with ESC.

Without the outer brackets the value returned by getchar() would be compared with the ESC character. If it was != ESC then the result is TRUE (non-zero) otherwise it is FALSE (zero). This TRUE or FALSE result (non-zero or zero) would then be fed into the inp_char variable. The final result of the condition is then based on whether or not inp_char now contains zero or non-zero! This would guarantee that the program didn't work as desired!

This type of condition is more elegant than the one that we used in the while loop of fig 1. It means that the statements within the loop do not need to check if inp_char is equal to ESC – if it is then the loop is automatically terminated. Although this new condition is more elegant than the old version, it is also a bit harder to understand.

The statements for the end of each repetition are executed at the end of each pass round the loop. If many statements are included they must be separated by commas. Note the use of the statement + +num_characters – this has the same effect as num 0characters = num _characters + 1 (it is a useful shorthand).

Some of the if statements in fig 2 look a bit different from before. The syntax used is:

```
if (condition)
statement1;
statement 2;
```

No curly brackets have been included. This means that statement1 is only executed if the condition is true. Statement2 is not included as part of the if statement. The basic idea is that if you want to make several statements dependent upon a condition you must use curly brackets to form a compound statement. If only one statement is to be dependent then the curly brackets are not needed – but you can include them if you want to.

The way that the code is indented in the example makes it look obvious what is happening. However, remember that the C compiler could not care less about the way that your code is indented. However you choose to indent the code, statement2 will always be executed if no curly brackets are used!

This principal applies to while and for loops also. For example:

```
while (condition)
statement1;
statement 2;
```

Because curly brackets have not been used, only statement1 is executed as part of the loop. Statement2 only happens after the loop has finished.

Most of the coding within the for loop uses some standard library functions to work out whether or not inp_char is alphabetic, a numeric digit etc. Each of these functions return TRUE or FALSE. Note the use of the == symbol when comparing inp_char with a space.

```
If (inp_char == ' ')
```

If a single = symbol were used it would mean: make inp_char equal to a space (Ascii code 32). If inp_char is then 0 (which it won't be – it will be 32!) the condition is FALSE, otherwise it is TRUE. The accidental use of = instead of == can cause newcomers to C a lot of frustration!

The only tricky bit involved in the loop is keeping track of the number of words that have been entered. It is made even more tricky when you try to define what you mean by a word and what kind of characters you expect to be used to separate words from each other. In fig 2 words are assumed to begin with an alphabetic character and are separated by spaces, tabs and newline characters.

Whenever a character that is a word separator is detected the within _a_word variable is set to FALSE, this indicates that we are not currently inside a word.

When any other character is detected nothing happens if within_a_word is TRUE.

If within_a_word is false it means we are possibly at the beginning of a new word. If the inp_char is alphabetic then the num_words variable is incremented since we really are at the beginning of a new word. The within_a_word variable is set to TRUE so that the next input character doesn't cause num_words to be updated again.

The printf function near the end of fig 2 illustrates another facility. The use of %6d as a print format tells the compiler to print the integer value right aligned in a 6 character field. This makes the printed values line up correctly underneath the headings that are output by the previous printf.

Exercise 2

1. No check has been made to stop num_characters (or any of the other integers) from exceeding MAX_INT. What would be the consequence of this happening? Change the code so that it cannot happen.
2. Try replacing the == in the check inp_char == ' ' by a single = symbol.
3. Make the program count the number of lines that you enter. The (ENTER) key on the keyoard enters a '\n' (newline) character.
4. Try printing some of the results with a print format that is too small (eg %1d).
5. Change the program so that word separators are any characters which are:

```
not alphabetic
and not one of:

'  (apostrophe)
-  (hyphen)
_  (underscore)
digits 0 through 0
```

Compiling and running

If the Digital C disk is on flp1_ and the program code is in a file named myprogram_c on flp2_ use the following to compile, link and run it. If error messages appear when you run the compiler you should edit the C program and compile it again.

```
exec_w flp1_cc (ENTER)
flp2_myprogram –p –m (ENTER)
exec_w flp1_cg (ENTER)
flp2_myprogram_exe flp2_myprogram
  –m –p –nc –d/flp1_. (ENTER)
exec_w flp2_myprogram_exe
```

Remember that Digital C source program files must end with _c.

ONE MAN'S SYSTEM

Neil Gordon trusts the QL with his health and education.

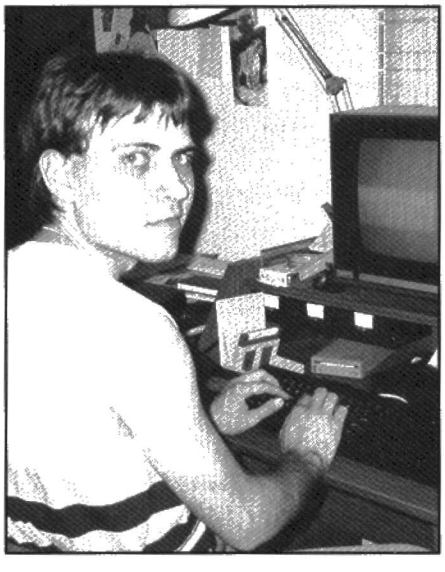
It is now eight years since I bought my first computer, a Sinclair ZX Spectrum, at the age of 13, back in 1982. At the time it was sufficient for my needs – learning abut computers and playing games! There was little opportunity at school to use computers, and there was little choice in home computers at the time.

A few years later, I wanted a more powerful micro. At this point, in 1984, the Sinclair QL was making its appearance, and it sounded like a wonderful machine. Several months later, once I had managed to get together sufficient money (part of which came from selling the Spectrum), I finally bought one. By then the dreaded kludge had gone, and the QL was fairly near completion. The keyboard was much better than the old Spectrum rubber one, and even now, after using many 'proper' keyboards, it still seems good. I am still using that machine, and have had no problems, except towards the end of the first year when I returned it under warranty for a faulty microdrive mechanism to be repaired.

The only leisure software available back then consisted of two text adventures (*West* and *Zkull*, both produced by Talent) and *Psion Chess*. I bought Zkull and the chess game from W H Smiths, since QL software was available in shops at the time. Now nearly everything has to be bought by mail order, but I have had no problems with such companies. These two games have probably been the cause of more intensive thought than any other software I have used! I am nearly always beaten by Psion Chess, while Zkull has yet to be completed. One benefit with the QL was that, since no one else had one, there was no problem at school, where not pirating software was deemed strange among my peers.

At first the basic QL hardware was sufficient for my limited requirements. I used a Saisho TV/Monitor for my vdu, a job which it is still happily fulfilling. Dixons supplied a suitable lead. In this first year, the absence of commercial software encouraged me to do more programming than I might otherwise have done. This time of exploring the QL's abilities was very enjoyable.

Slowly my software collection increased, though generally only with games. I bought *QL User* from the start, and frequently copied the listings published in it, and later in *QL World*. To simplify finding and loading particular programs on a microdrive cartridge, which often contained 40 or more files, I wrote a program to list all the files on a cartridge and allow their selection by at most two key presses.



The author with his compact QL set-up.

At the age of 18 I started a joint mathematics and computer science course at Hull University. In the first year we learnt Pascal and had several programming assignments on the departmental IBM compatibles. My experience with *Superbasic* proved to be very useful since it has many similarities with Pascal. I bought a Pascal compiler (by Computer One) which allowed me to try out programs at home. The programming assignments required two reports to be submitted: an initial one, and then a final one with the completed program. I bought a Brother HR-5 thermal printer which provided high enough quality output to be suitable for the reports. It also allowed me to take printed listings to copy onto the Departments micros.

By this stage, I was reaching the limitations of a standard QL. Quill with microdrives is not very inspiring! Miracle Systems' 512K *Expanderam* was available fairly cheaply at the time because of the fairly low cost of ram chips back then, so I bought one from Byteback, together with the *QFlash* toolkit rom which gave me a 'permanent' ramdisk.

This system proved adequate for writing a summer vacation essay about computers! In fact Quill has been a very good word processor, capable of all the tasks I have put it to, which included writing cvs.

I was given a disk drive system for Christmas in 1988. This consisted of a Cumana interface together with a single, double sided, double density drive. Strong Computer Systems supplied the system, which included 10 disks. This improved things a great deal, and the entire system is excellent.

Over the past five years I have bought

```
10 REMark LISTING 2 - 'Skeleton' program to produce blood glucose
20 REMark results analyser from stats program.
30 REMark Load stats first, then merge this with it.
210 PRINT #titlewindow;'BLOOD GLUCOSE STATISTICS.'
300  defaultdevice$='mdv1_glucose_'
310  maxval=25
320  minval=0
3465  PRINT #datawindow;TO 9;'Blood Glucose against days.'
3475  PRINT #datawindow;TO 9;'Lines for Max,Optimal & Min.'
3477  PRINT #datawindow;\\\'m'\'m'\'o'\'l'\'/'\'l'
3681  REMark Draw lines for optimum max,normal&min blood glucose levels
3682  RESTORE 3688
3683  FOR x=1 TO 3
3684   READ y
3685   LINE #datawindow,leftorigin,(y-minval)*yscale TO 140,(y-minval)*yscale
3686   LINE #datawindow,leftorigin,(y-minval)*yscale+hiyposn TO 140,(y-minval)*yscale+hiyposn
3687  END FOR x
3688  DATA 3.5,5,8
```

```
100 REMark LISTING 1.
110 REMark Neil Gordon. 1990
120 REMark General statistical analysis of
discrete data.
130 REMark Holds one months figures, and ca
lculates
140 REMark the mean and modal values, both
overall
150 REMark and by fields. (NB 7 fields)
160 REMark Best keeping to 2 digit data for
 table layout.
170 REMark Ie. Integers in range -99 to 99.
180 init
190 init_display
200 REMark title
210 PRINT #titlewindow;'STATISTICAL ANALYSE
R.'
220 displayalldata
230 menu
240 STOP
250 DEFine PROCedure init
260 REMark initialise data & main data stru
cture.
270  RESTORE
280  REMark default values
290  defaultfile$='No Name'
300  defaultdevice$='mdv1_'
310  maxval=99
320  minval=-99
330  nullval=100:REMark used to signify 'no
' data value.
340  xmin=2:REMark left hand position of da
ta for printing
350  DIM daysdata(31,7):REMark 1 month,7 te
sts a day.
360  resetdata
370 END DEFine init
380 DEFine PROCedure resetdata
390  REMark initialise to nullval - program
 ignores such values
400  FOR x=1 TO 31
410   FOR y=1 TO 7
420    daysdata(x,y)=nullval
430   END FOR y
440  END FOR x
450 END DEFine
460 DEFine PROCedure init_display
470  LOCal wwidth,height,xpos,ypos,n
480  MODE 4
490  OPEN_NEW #3,scr_512x256a0x0
500  PAPER #3,4
510  CLS #3
520  CLOSE #3
530  READ numberofwindows
540  FOR n=3 TO numberofwindows+2
550   READ wwidth,height,xpos,ypos
560   OPEN_NEW #n,con_
570   WINDOW #n,wwidth,height,xpos+5,ypos+5
580   PAPER #n,0
590   CLS #n
600   WINDOW #n,wwidth,height,xpos,ypos
610   PAPER #n,7
620   INK #n,0
630   CLS #n
640  NEXT n
650  REMark window data
660   DATA 6:REMark number of windows
670   datawindow=3
680   DATA 302,180,20,30
690   menuwindow=4
700   DATA 90,70,355,30
710   inputwindow=5
720   DATA 302,11,20,222
730   resultswindow=6
740   DATA 130,130,355,110
750   statuswindow=7
760   DATA 250,11,25,10
770   titlewindow=8
780   DATA 150,11,330,10
790  REMark end window data
800 END DEFine init_display
810 DEFine PROCedure menu
820  LOCal ch$,choice
830  REPeat loop
840   PRINT #statuswindow;'Present File is
: ';defaultfile$
850   CLS #menuwindow
860   PRINT #menuwindow,'1 - Load'\'2 - Sav
e'\'3 - Input Data'\'4 - Results'\'5 - Zap'
\'6 - Quit'\'7 - Graph'
870   REPeat chloop
880    ch$=INKEY$(-1)
890    IF ch$ INSTR "1234567" <> 0 THEN EXI
T chloop
900   END REPeat chloop
910   CLS #menuwindow
920   choice=ch$
930   SELect ON choice
940    =1:loader
950    =2:saver
960    =3:adder
970    =4:results
980    =5:zapper
990    =6:quiter
1000    =7:graphs
1010   END SELect
1020   CLS #inputwindow
1030  END REPeat loop
1040 END DEFine menu
1050 DEFine PROCedure adder
1060 REMark input data
1070  LOCal daynum,testnumber,inputstring$,
inputvalue
1080  REPeat adderloop
1090   INPUT #inputwindow;'Day - ';inputstr
ing$
1100   IF inputstring$='' THEN EXIT adderlo
op
1110   daynum=inputstring$
1120   IF daynum<1 OR daynum>31 THEN GO TO
1090
1130   INPUT #inputwindow;'Test No - ';test
number
1140   IF testnumber<1 OR testnumber>7 THEN
 GO TO 1130
1150   INPUT #inputwindow;'Data Value - ';i
nputstring$
1160   IF inputstring$='' THEN
1170    datavalue=nullval
1180   ELSE
1190    datavalue=inputstring$
1200    IF datavalue<minval OR datavalue>ma
xval THEN GO TO 1150
1210   END IF
1220   daysdata(daynum,testnumber)=datavalu
e
1230   displaydata (daynum),(testnumber)
1240  END REPeat adderloop
```

```
1250  CLS #inputwindow
1260  CLS #resultswindow
1270 END DEFine adder
1280 DEFine PROCedure printdata(x)
1290  REMark print data in red if <0, in bl
ack if >=0
1300  REMark x=nullval means ignore
1310  IF x=nullval THEN REturn
1320  IF x<0 THEN
1330   INK #datawindow,2
1340   PRINT #datawindow;-x
1350   INK #datawindow,0
1360  ELSE
1370   PRINT #datawindow;x
1380  END IF
1390 END DEFine printdata
1400 DEFine PROCedure displayalldata
1410  LOCal x,y,num
1420  CLS #datawindow
1430  PRINT #datawindow;'Date.        Time.
       Date.        Time.'
1440  AT #datawindow,17,31
1450  PRINT #datawindow;' (Negative=red)'
1460  FOR num=1 TO 2
1470   FOR tst=1 TO 7
1480    AT #datawindow,1,xmin+25*(num=2)+ts
t*3;
1490    PRINT #datawindow;tst
1500   END FOR tst
1510  END FOR num
1520  FOR num=1 TO 31
1530   x=xmin
1540   y=num
1550   IF num>16 THEN x=xmin+25:y=y-16
1560   AT #datawindow,y+1,x-1
1570   PRINT #datawindow;num
1580   FOR tst=1 TO 7
1590     AT #datawindow,y+1,x+(tst*3)
1600     printdata daysdata(num,tst)
1610   END FOR tst
1620  END FOR num
1630  REMark draw table
1640  LINE #datawindow,10,0 TO 10,89.5 TO 0
,89.5
1650  LINE #datawindow, 10,89.5 TO 63,89.5
1660  LINE #datawindow, 63,0 TO 63,89.5 TO
124,89.5
1670  LINE #datawindow, 71,0 TO 71,89.5
1680 END DEFine displaydata
1690 DEFine PROCedure displaydata(no,tstno)

1700 REMark display one item of data
1710  LOCal x,y
1720  x=xmin+tstno*3
1730  y=no+1
1740  IF no>16 THEN y=y-16:x=x+25
1750  AT #datawindow,y,x
1760  PRINT #datawindow;'  '
1770  AT #datawindow,y,x
1780  printdata daysdata(no,tstno)
1790 END DEFine
1800 DEFine PROCedure saver
1810  LOCal devicename$,filename$,name$,num
ber,tstnum
1820  INPUT #inputwindow;'Save Data - Sure?
 ';ch$
1830  IF NOT(ch$=='y') THEN RETurn
1840  PRINT #inputwindow;'File Device(defau
lt-';defaultdevice$;') - ';
1850  INPUT #inputwindow;devicename$
1860  IF devicename$='' THEN
1870   devicename$=defaultdevice$
1880  ELSE
1890   defaultdevice$=devicename$
1900  END IF
1910  INPUT #inputwindow;'File Name  - ';fi
lename$
1920  IF filename$='' THEN
1930   filename$=defaultfile$
1940  ELSE
1950   defaultfile$=filename$
1960  END IF
1970  name$=devicename$&filename$
1980  PRINT #menuwindow;'SAVING...'
1990  DELETE name$
2000  OPEN_NEW #15,name$
2010  FOR number=1 TO 31
2020   FOR tstnum=1 TO 7
2030    PRINT #15;daysdata(number,tstnum)
2040   END FOR tstnum
2050  END FOR number
2060  CLOSE #15
2070 END DEFine saver
2080 DEFine PROCedure loader
2090  LOCal ch$,devicename$,name$,filename$
,number,tstnum
2100  INPUT #inputwindow;'Loader - Sure? ';
ch$
2110  IF NOT(ch$=='y') THEN RETurn
2120  PRINT #inputwindow;'File Device(defau
lt-';defaultdevice$;') - ';
2130  INPUT #inputwindow;devicename$
2140  IF devicename$='' THEN
2150   devicename$=defaultdevice$
2160  ELSE
2170   defaultdevice$=devicename$
2180  END IF
2190  INPUT #inputwindow;'File Name  - ';fi
lename$
2200  defaultfile$=filename$
2210  name$=devicename$&filename$
2220  PRINT #menuwindow;'LOADING...'
2230  OPEN_IN #15,name$
2240  FOR number=1 TO 31
2250   FOR tstnum=1 TO 7
2260    INPUT #15;daysdata(number,tstnum)
2270   END FOR tstnum
2280  END FOR number
2290  CLOSE #15
2300  displayalldata
2310  CLS #resultswindow
2320 END DEFine loader
2330 DEFine PROCedure zapper
2340 REMark remove data
2350  LOCal ch$,num,tstnum
2360  INPUT #inputwindow;'Zap data - Sure?
';ch$
2370  IF NOT(ch$=='y') THEN RETurn
2380  resetdata
2390  defaultfile$='No file'
2400  CLS #resultswindow
2410  displayalldata
2420 END DEFine zapper
2430 DEFine PROCedure quiter
2440 REMark quit program
2450  LOCal ch$
2460  INPUT #inputwindow;'Quit Program - Su
re? ';ch$
```

```
2470  IF NOT(ch$=='y') THEN RETurn
2480  FOR n=3 TO numberofwindows
2490   CLOSE #n
2500  END FOR n
2510  NEW
2520 END DEFine
2530 DEFine PROCedure results
2540  LOCal tn,n,mean,totalitems,temp,varia
nce
2550   PRINT #menuwindow;'CALCULATING'\'RES
ULTS...'
2560  maxlevel=maxval-(minval*(minval<0))
2570  REMark first find mean and mode by fi
eld & overall.
2580  DIM meanval(7,2):REMark ,1 is total,
,2 is no of data items
2590  DIM modalval(7,maxlevel),overallmode(
maxlevel)
2600  mean=0:totalitems=0
2610  FOR tn=1 TO 7
2620   FOR n=1 TO 31
2630    IF daysdata(n,tn)<>nullval THEN
2640     temp=daysdata(n,tn)
2650     meanval(tn,1)=meanval(tn,1)+temp
2660     mean=mean+temp
2670     increment totalitems
2680     increment meanval(tn,2)
2690     IF (temp<0 AND minval<0) THEN
2700      increment modalval(tn,maxval-temp
)
2710      increment overallmode(maxval-temp
)
2720     ELSE
2730      increment modalval(tn,temp)
2740      increment overallmode(temp)
2750     END IF
2760    END IF
2770   END FOR n
2780  END FOR tn
2790  CLS #resultswindow
2800  PRINT #resultswindow;'Mean and Mode b
y test'\'times :'
2810  FOR tn=1 TO 7
2820   modal=0:occurances=0
2830   FOR n=1 TO maxlevel
2840    IF modalval(tn,n)>occurances THEN
2850     occurances=modalval(tn,n)
2860     modal=n
2870    END IF
2880   END FOR n
2890   IF meanval(tn,2)=0 THEN meanval(tn,2
)=1
2900   meanval(tn,1)=(meanval(tn,1)/meanval
(tn,2))
2910   round meanval(tn,1)
2920   IF modal>maxval THEN modal=-modal+ma
xval
2930   PRINT #resultswindow;tn;TO 3;meanval
(tn,1);TO 7;modal
2940  END FOR tn
2950  IF totalitems=0 THEN totalitems=1
2960  mean=mean/totalitems
2970  REMark round mean to nearest integer.

2980  temp=mean
2990  round temp
3000  PRINT #resultswindow;'Overall Mean=';
temp
3010  REMark now get the standard deviation
.
3020  variance=0
3030  FOR n=1 TO 31
3040   FOR tn=1 TO 7
3050    IF daysdata(n,tn)<>nullval THEN
3060     variance=variance+(daysdata(n,tn)^
2)
```

two joysticks. One, a Voltmace Delta QL, eventually broke; however, it had suffered valiantly at the hands of myself and assorted others who have enjoyed the wonders of Sinclair's QL. Unfortunately, this is no longer available. My present joystick, a Wiz Card, is actually more of a cursor pad, and I tend to use it more as a mouse substitute, a job for which it is eminently suitable.

As I have required new languages for my course, I have bought the necessary compilers. This is one of the major benefits of the QL. Such compilers are available for most of the major languages, and are fairly cheap compared to those on other computers. I use the Prospero Fortran compiler, again bought from Strong Computer Systems. It is very good, but requires the supplied eprom to be present. This means I cannot use the Qflash eprom with it. This is unfortunate, since it would be useful at times to use a ramdisk rather than the disk drive with the compiler. It is for this reason that I have not invested in a mouse, since I prefer not to continually have to fit and remove such eproms. I considered one of the multi-rom adapters, but they seemed expensive.

I tried some assembly programming, but have only written fairly simple programs so far, such as adding new functions, which is so easy on the QL.

Last year I attended one of the Northern Computer Shows organised by Sector Software, which was very interesting. I am using the *Spellbound* spelling checker I bought there to check this article as I write it! Any spelling errors are due to my forgetting to switch it on, rather than any deficiencies in the software.

One problem with the hardware I have collected is the number of bulky, hot and noisy power adapters they require. I have alleviated the problems somewhat by fitting a small shelf on the back of my computer desk where they now reside.

Being diabetic, I keep a daily record of my blood glucose levels. I have finally begun to computerise these records, which allows me to get statistical data about this information. This is very useful since trying to see patterns in a table of figures is difficult, and calculating average values and so on is very time consuming. None of the Psion software seemed quite right for this job, since I wanted a full month's figures on screen at once, and to get a plot of the values together with a moving average. For this reason I wrote a SuperBasic program to do the job. I have included a generalised listing of this **(listing 1)**, since it may be useful for others, being suitable for any integer data input on a daily basis. I have also included a short program **(listing 2)** to merge with this to produce the specialised version. To create the latter version, first load the general version, then merge the short program with it. The listings are fairly self explanatory, and contain comments.

Of all the languages I now know, Superbasic is probably one of the best. It is useful in the same way as the shell language found on Unix systems; for example, to write prototypes. Unlike that, it is fairly friendly (in terms of error messages, many is the hour I have spent searching for an error which was caused by using a colon instead of a semi-colon. The only real problem with Superbasic is its speed, but this can be improved by using one of the Basic compilers available.

Since speed is not always that important, I have not yet bought one of these. It is surprising just how many similarities there are between the QL and a Unix system, as in the use of pipes and multi-tasking.

The QL was and continues to be an excellent micro for students and anyone requiring a useful computer. Just how good it was is seen from the fact that although I would quite like a better-supported micro, there is at the present time no micro more suited to my needs than my trusty QL.

Finally, I have passed my finals, getting a First Class Honours degree, and will hopefully continue for either a Masters or a PHD, and some of the credit must go to my QL which was a useful tool and teacher!

```
3070    END IF
3080   END FOR tn
3090  END FOR n
3100  variance=(variance/totalitems)-(mean^
2)
3110  PRINT #resultswindow;'Standard Deviat
ion'
3120 -PRINT #resultswindow;' (to mean)    =';
(INT(SQRT(variance)*100))/100
3130  modal=0:occurances=0
3140  FOR n=1 TO maxlevel
3150   IF overallmode(n)>occurances THEN
3160    occurances=overallmode(n)
3170    modal=n
3180   END IF
3190  END FOR n
3200   IF modal>maxval THEN modal=-modal+ma
xval
3210  PRINT #resultswindow;'Overall Mode=';
modal
3220 END DEFine results
3230 DEFine PROCedure round(x)
3240  IF x-INT(x)>.5 THEN
3250   x=INT(x)+1
3260  ELSE
3270   x=INT(x)
3280  END IF
3290 END DEFine increment
3300 DEFine PROCedure increment(x)
3310  x=x+1
3320 END DEFine increment
3330 DEFine PROCedure graphs
3340 REMark display graph of data, and calc
ulate moving average.
3350  LOCal ox,oy,x,y,n,tn,xstep,leftorigin
,hiyposn,mvavpar,loopn,mvavch$,yscale,gmaxv
al,gminval
3360  REMark get moving average parameter.
3370  INPUT #inputwindow;'No. of days for m
oving average?: ';mvavch$
3380  IF mvavch$='' THEN
3390   mvavpar=-1
3400  ELSE
3410   mvavpar=mvavch$
3420   IF mvavpar<1 OR mvavpar>31 THEN GO T
O 3370
3430  END IF
3440  CLS #datawindow
3450  CLS #inputwindow
3460  INK #datawindow,4
3470  IF mvavpar<>-1 THEN PRINT #datawindow
;TO 9;'Graph with ';mvavpar;'-day moving av
erage.'
3480  AT #datawindow,17,2:PRINT #datawindow
;minval
3490  AT #datawindow,9,2:PRINT #datawindow;
maxval
3500  AT #datawindow,8,2:PRINT #datawindow;
minval
3510  AT #datawindow,0,2:PRINT #datawindow;
maxval
3520  AT #datawindow,9,5:PRINT #datawindow;
'1  2  3  4  5  6 7  8  9 10 11 12 13 14 15
 16'
3530  AT #datawindow,10,17:PRINT #datawindo
w;' (Date)'
3540  AT #datawindow,17,5:PRINT #datawindow
;'17 18 19 20 21 22 23 24 25 26 27 28 29 30
 31'
3550  INK #datawindow,0
3560  hiyposn=50
3570  yscale=50/(maxval-minval):REMark y sc
aling factor
3580  leftorigin=12
3590  REMark axis
3600  INK #datawindow,4
3610  REMark y axis
3620  LINE #datawindow,leftorigin,0 TO left
origin,100
3630  REMark dividing line
3640  LINE #datawindow,leftorigin,hiyposn T
O 140,hiyposn
3650  REMark upper x axis(y=0)
3660  LINE #datawindow,leftorigin,-minval*y
scale+hiyposn TO 140,-minval*yscale+hiyposn

3670  REMark lower x axis (y=0)
3680  LINE #datawindow,leftorigin,-minval*y
scale TO 140,-minval*yscale
3690  INK #datawindow,0
3700  xstep=1
3710  oy=nullval
3720  FOR n=1 TO 31
3730   IF (n=17 OR n=1) THEN
3740    ox=leftorigin
3750    x=ox
3760   END IF
3770   FOR tn=1 TO 7
3780    x=x+xstep
3790    y=daysdata(n,tn)
3800    IF oy=nullval AND daysdata(n,tn)<>n
ullval THEN oy=y:ox=x
3810    IF daysdata(n,tn)<>nullval THEN
3820     LINE #datawindow;ox,(oy-minval)*ys
cale+hiyposn*(n<17) TO x,(y-minval)*yscale+
hiyposn*(n<17)
3830     ox=x:oy=y
3840    END IF
3850   END FOR tn
3860  END FOR n
3870  REMark calculate & draw moving averag
e
3880  IF mvavpar<>-1 THEN
3890   mvavpar=mvavpar/2+.5
3900   INK #datawindow,2
3910   ox=leftorigin+mvavpar*7-3.5*xstep
3920   oy=nullval
3930   FOR loopn=mvavpar TO 32-mvavpar
3940    movingaverage=0
3950    numberofitems=0
3960    FOR n=loopn-mvavpar+1 TO loopn+mvav
par-1
3970     FOR tn=1 TO 7
3980      IF daysdata(n,tn)<>nullval THEN
3990       movingaverage=movingaverage+days
data(n,tn)
4000        increment numberofitems
4010       END IF
4020     END FOR tn
4030    END FOR n
4040    x=loopn*7+leftorigin-16*7*(loopn>=1
7)-3.5*xstep
4050    IF numberofitems=0 THEN
4060     y=oy
4070    ELSE
4080     y=movingaverage/numberofitems
4090    END IF
4100    IF oy=nullval THEN ox=x:oy=y
4110    IF numberofitems<>0 THEN
4120    LINE #datawindow;ox,(oy-minval)*ysc
ale+hiyposn*(loopn<17) TO x,(y-minval)*ysca
le+hiyposn*(loopn<17)
4130     ox=x:oy=y
4140    END IF
4150    IF INT(loopn)=16 THEN ox=leftorigin
4160   END FOR loopn
4170   INK #datawindow,0
4180  END IF
4190  PRINT #menuwindow;'ANY KEY TO'\'CONTI
NUE...'
```

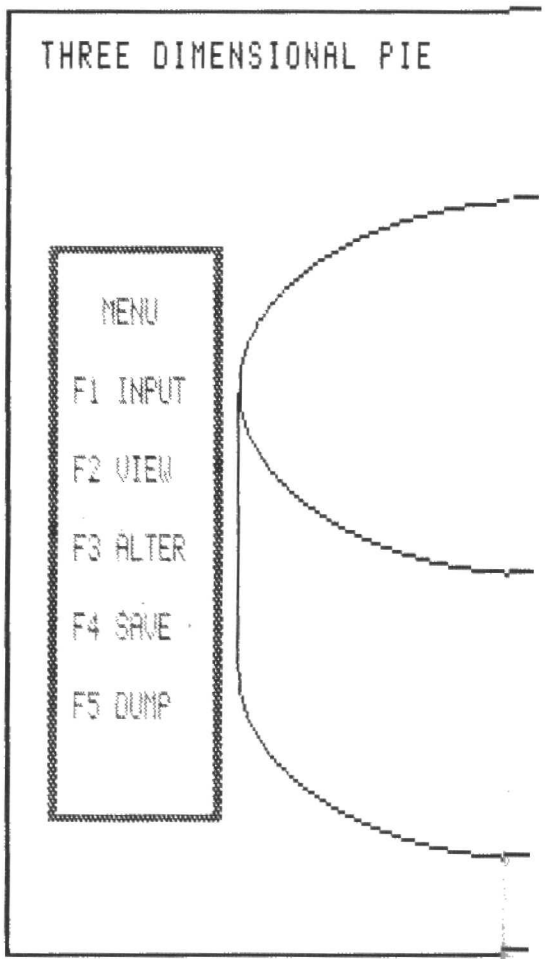
```
10 MODE 4
20 WINDOW 512,256,0,0
30 PAPER 0:INK 1:CLS
40 initialise
50 start
60 menu
3399 :
3400 DEFine PROCedure presuf
3410 CLS#4
3420 WINDOW#0,436,24,39,11
3430 INPUT#0;"Prefix: ";pre$
3440 CLS#0
3450 INPUT#0;"Suffix: ";suf$
3460 CLS#0
3470 END DEFine
3599 :
3600 DEFine PROCedure change_col
3610 IF fink<>2
3620  IF fink=4 AND seg<>3*INT(seg)/seg+1
3630   fink=184:sink=248
3640  ELSE
3650   fink=2:sink=208
3660  END IF
3670 ELSE
3680  fink=4:sink=224
3690 END IF
3700 END DEFine
3799 :
3800 DEFine PROCedure change_bw
3810 IF fink<>248
3820  IF fink=120 AND seg<>3*INT(seg)/seg+1
3830   fink=0:sink=0
3840  ELSE
3850   fink=248:sink=248
3860  END IF
3870 ELSE
3880   fink=120:sink=120
3890 END IF
3900 END DEFine
3999 :
4000 DEFine PROCedure fill_pie
4010 tempang=2*PI
4020 INK fink
4030 ang=ang1:angle:x1=x:y1=y
4040 IF ang1<PI AND ang2>PI:tempang=ang2:ang2=PI
4050 ang=ang2:angle:x2=x:y2=y
4060 x3=70+x1:x4=70+x2
4070 IF ang1<=PI AND ang2<=PI
4080  IF x3<70 AND x4<70:x3=70
4090  IF x3>70 AND x4>70:x4=70
4100  FOR x=x3 TO x4+.2 STEP -.2
4110   y3=45+height+SQRT((1-((x-70)/xrad)^2)*(yrad
^2))
4120   y4=45+height+(x-70)*TAN(ang1)
4130   y5=45+height+(x-70)*TAN(ang2)
4140   IF (x1+70>70 AND x2+70>70 AND x<x2+70) OR (
x1+70<70 AND x2+70<70 AND x>x1+70)
4150    LINE x,y4 TO x,y5
4160   ELSE
4170    IF x>70
4180     LINE x,y3 TO x,y4
4190    ELSE
4200     LINE x,y3 TO x,y5
4210    END IF
4220   END IF
4230  NEXT x
4240 END IF
4250 IF tempang<>2*PI:ang2=tempang:tempang=ang1:an
g1=PI
4260 ang=ang1:angle:x1=x:y1=y
```

A 3-D

Geoff Wicks improves on Easel to cut pie charts to fit dtp documents or other uses.

THREE Dimensional Pie is a program which allows you to design pie charts for display on the screen or for printing to a dot matrix printer. A special feature of the program is that the charts can easily be entered into documents produced by desktop publishing programmes. Pie charts can, of course, be produced by *Easel*, but they are rather lifeless. Three Dimensional Pie aims to produce better pies than Easel and with one exception — the program does not support highlighting — is more versatile than Easel.

When you load 3DPie the basic pie appears in the centre of the screen. To the left is a menu window and to the right a status window. Practically all the menus



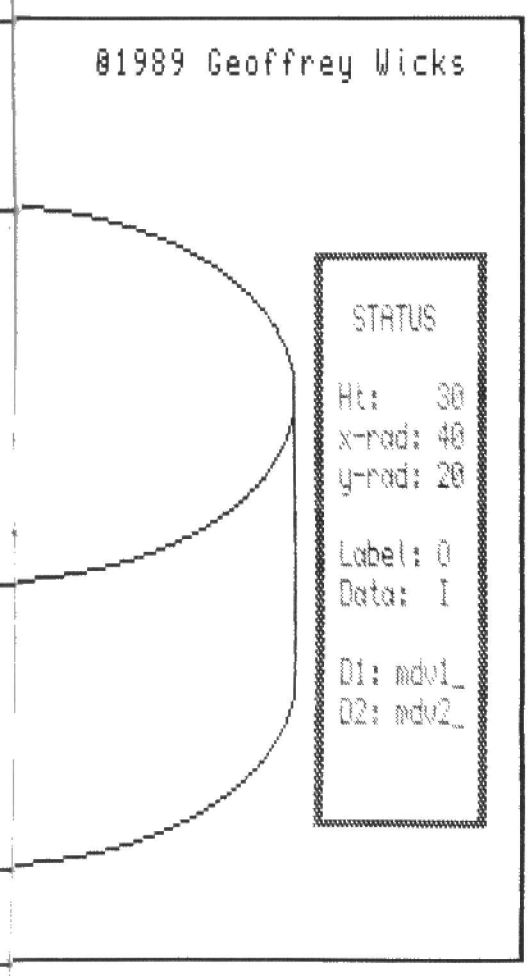
PIE

make use of the function keys, but if you have an averson to this sort of menu, don't panic. Mnemonics are also used, so you can get the same result by pressing the initial letter of the menu item. The mnemonics work with both upper and lower case leters — one of the advantages of the SELect ON keyword.

The main menu F1 (or "I" or "i") is for Input. Inputting can be done in one of three ways and whenyou opt for input you are presented with the input menu:

F1 (or "D" or "d") is for Demo. It uses data contained in lines 9940. You could change these lines to enter your own data. Line 9890 is the number of slices in thepie (maximum 12) and the title line 9900 the labels, and line 9910 the data. Line 9930 is for prefix and suffix if required.

F2 (or "M" or "m") is for Manual. Choosing this option opens a window in the centre of the screen and you are invited to enter the title of your pie. The screen is cleared and you can then enter your first label followed by the data for that label. This process is repeated for a maximum of 12 slices. Should you require




```
4270 ang=ang2:angle:x2=x:y2=y
4280 x3=70+x1:x4=70+x2
4290 IF ang1>=PI AND ang2>=PI
4300  IF x3<70 AND x4<70:x4=70
4310  IF x3>70 AND x4>70:x3=70
4320  FOR x=x3+.2 TO x4 STEP .2
4330   y3=45+height-SQRT((1-((x-70)/xrad)^2)*(yrad
^2))
4340   y4=45+height+(x-70)*TAN(ang1-PI)
4350   y5=45+height+(x-70)*TAN(ang2-PI)
4360   IF (x1+70<70 AND x2+70<70 AND x>x2+70) OR (
x1+70>70 AND x2+70>70 AND x<x1+70)
4370    LINE x,y4 TO x,y5
4380   ELSE
4390    IF x<70
4400     LINE x,y4 TO x,y3
4410    ELSE
4420     LINE x,y5 TO x,y3
4430    END IF
4440   END IF
4450  NEXT x
4460  IF side$="Outline"
4470   INK sink
4480   FOR x=x1+70.2 TO x2+69.8 STEP .2
4490    y3=45+height-SQRT((1-((x-70)/xrad)^2)*(yra
d^2))
4500    LINE x,y3 TO x,y3-2*height
4510   NEXT x
4520  END IF
4530 END IF
4540 IF tempang<>2*PI:ang1=tempang:tempang=2*PI
4550 IF filcol$="Colour ":change_col:ELSE :change_
bw
4560 INK 7
4570 END DEFine
4599 :
4600 DEFine PROCedure move_label
4610 Lab$="Manual ":Dat$="Manual "
4620 CLS #4
4630 FOR n = 1 TO seg
4640  FOR o = 1 TO 2
4650   IF o=1
4660    x=lx%(n):y=ly%(n)
4670   ELSE
4680    x=dx%(n):y=dy%(n)
4690   END IF
4700   INK 2:CURSOR x,y,0,0:print_ld
4710   OVER 1
4720   REPeat move_ld
4730    key=CODE(INKEY$(-1))
4740    SELect ON key
4750     = 192
4760      CURSOR x,y,0,0:INK 0:print_ld:x=x-1:INK
2:CURSOR x,y,0,0:print_ld
4770     = 200
4780      CURSOR x,y,0,0:INK 0:print ld:x=x+1:INK
2:CURSOR x,y,0,0:print_ld
4790     = 208
4800      CURSOR x,y,0,0:INK 0:print_ld:y=y+1:INK
2:CURSOR x,y,0,0:print_ld
4810     = 216
4820      CURSOR x,y,0,0:INK 0:print_ld:y=y-1:INK
2:CURSOR x,y,0,0:print_ld
4830     = 32:EXIT move_ld
4840     = 27: menu
4850    END SELect
4860   END REPeat move_ld
4870   OVER 1:INK 7
4880   CURSOR x,y,0,0
4890   IF o=1
4900    PRINT Label$(n):lx%(n)=x:ly%(n)=y
```

```
4910    ELSE
4920     PRINT pre$;num(n);suf$:dx%(n)=x:dy%(n)=y
4930    END IF
4940   NEXT o
4950 NEXT n
4960 view_pie
4970 END DEFine
4999 :
5000 DEFine PROCedure print_ld
5010 IF o=1
5020  PRINT Label$(n)
5030  PRINT#4,"LABEL ";n;": ";Label$(n),"x: ";x,"y
: ";y
5040 ELSE
5050  PRINT pre$;num(n);suf$
5060  PRINT#4,"DATA ";n;": ";pre$;num(n);suf$,"x:
";x,"y: ";y
5070 END IF
5080 END DEFine
5199 :
5200 DEFine PROCedure design
5210 WINDOW#0,260,146,129,70:CLS#0:BORDER#0,2,248
5220 AT#0,1,8:PRINT#0,"BORDER",,borcol$
5230 AT #0,2,8:PRINT#0,"EFFECT",,effect$
5240 AT#0,3,8:PRINT#0,"SIDE/BASE",side$
5250 AT#0,4,8:PRINT#0,"FILL",,filcol$
5260 AT#0,6,8:PRINT#0,"LABELS",,Lab$
5270 AT#0,7,8:PRINT#0,"DATA",,Dat$
5280 AT#0,9,8:PRINT#0,"1st DRIVE",d1$
5290 AT#0,10,8:PRINT#0,"2nd DRIVE",d2$
5300 INK#0,7:AT#0,12,5:PRINT#0,"PRESS INITIAL LETT
ER TO CHANGE"\"              'ESC' TO RETURN":INK#0,
4
5310 REPeat loop
5320  key=CODE(INKEY$(-1))
5330  SELect ON key
5340   = 27:menu
5350   = 81,113:STOP
5360   = 66,98
5370    IF borcol$="On ":borcol$="Off":borcol=0:EL
SE borcol$="On ":borcol=7
5380    AT#0,1,8:PRINT#0,"BORDER",,borcol$
5390    BORDER 1,borcol
5400   = 69,101
5410    IF effect$="Solid ":effect$="Shadow":ELSE
:effect$="Solid "
5420    AT #0,2,8:PRINT#0,"EFFECT",,effect$
5430   = 83,115
5440    IF side$<>"Outline"
5450     IF side$="Stipple"
5460      side$="Solid  ":bcol=7
5470     ELSE
5480      side$="Outline":bcol=0
5490     END IF
5500    ELSE
5510     side$="Stipple":bcol=248
5520    END IF
5530    AT#0,3,8:PRINT#0,"SIDE/BASE",side$
5540   = 70,102
5550    IF filcol$<>"Outline"
5560     IF filcol$="Colour "
5570      filcol$="B/W    ":fink=248
5580     ELSE
5590      filcol$="Outline"
5600     END IF
5610    ELSE
5620     filcol$="Colour ":fink=2:sink=208
5630    END IF
5640    AT#0,4,8:PRINT#0,"FILL",,filcol$
5650   = 76,108
5660    Datlab$=Lab$:datlab:Lab$=Datlab$
```

less than 12 slices, pressing Enter when you are asked for the label will allow you to leave the sequence. (If you wish to have four segments, enter the label and data for each of the segments but, when asked to input the fifth label, just press enter).

F3 (or "I" or "i") is for Import. This option allows you to import data from the Psion programs. There are two import routines built into 3DP. The program first assumes that you are importing from Easel and tests for this. If the import is not successful it is aborted and a second routine is tried which assumes you are importing from *Abacus* by column. The importing routines have been kept as simple as possible and may not work in every case. If your attempted import fails, use the manual option which is simple and quick.

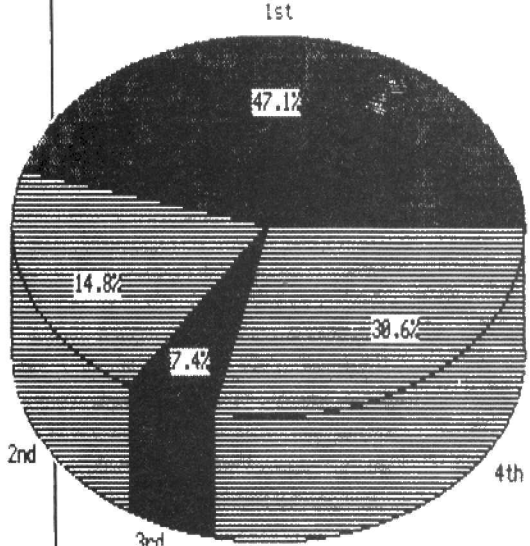
F2 (or "A" or "a") is for Alter. This allows you to change the design of the pie. On choosing this option you are presented with the alter menu:

F1 (or "H" or "h") to alter the Height.
F2 (or "X" or "x") to alter the X-radius.
F3 (or "Y" or "y") to alter the Y-radius.

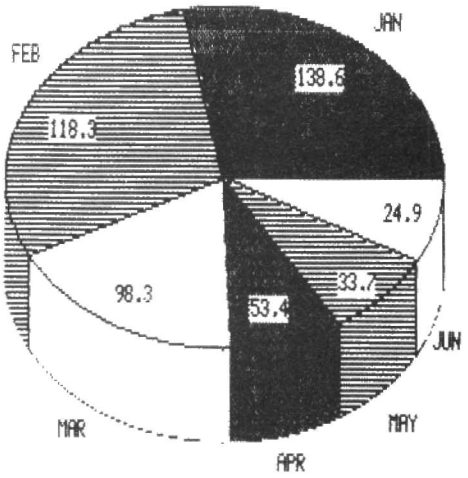
These three options all work in the same way. You alter the value by means of the up and down cursor keys. The altered value can be followed in the status window. When you have the value you require you press Space and the pie is redrawn using your new values. The permitted ranges are:

Height: 0–30
X-rad: 10–50
Y-rad: 5–35

You will need to experiment to get the best values for your pie. Themore "elliptical" your ellipse is, the greater theimpression of depth. Unfortunately although your pie will be, mathematically speaking, correctly sliced, it may not look right. For example a 10% slice may look larger than a 15% slice. The nearer to a circle your



Average gas consumption per quarter.



Average gas consumption per month.

ellipse is, the less distortion there is. It is a question of choosing the best compromise. If you are disappointed with the design in out line mode, persevere. Often the effect is much better when the pie is filled in. It also helps if you keep the height small.

F4 (or "D" or "d") is for Design. When you choose this option a window opens in the centre of the screen in which the Design Menu is displayed. The menu works entirely by mnenomics.

Border switches the border on and off. You will see this happening as you press the key. If you intend to use your pie in a desktop publishing program, you may find it better to switch the border off.

Effect toggles between Solid and Shadow. Solid draws a complete pie. Shadow gives the effect of a floating circle with its shadow underneath.

Side/Base toggles between Outline, Stipple and Solid. Outline uses the same colours for the base and the sides as fill (see below). Stipple colours them in a black and white stipple and Solid in white.

Fill toggles between Outline, Colour and B/W. Outline is the default mode and draws the pie in outline form. Colour fills in the pie in colour and B/W in black and white. The former gives a better result if you want to view the pie on the screen or to photograph it; the latter is better for use in desktop publishing.

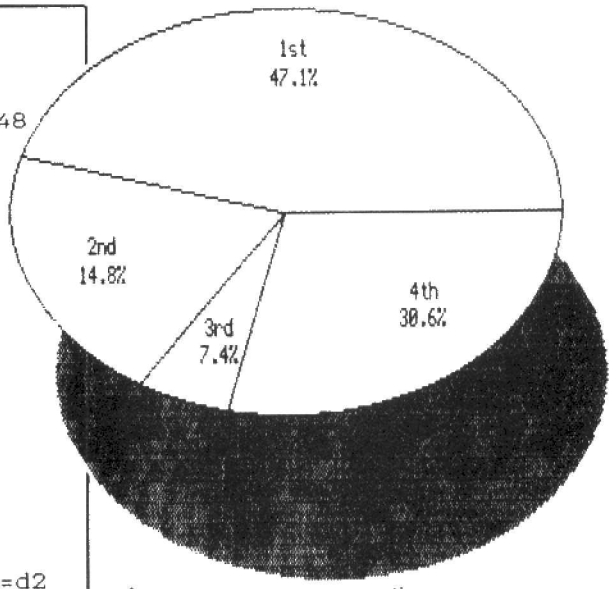
Becuse some unusual shapes have to be filled in, the fill routine is the slowest part of the program. You are advised to design your pie first in outline form and add the colour as the last feature. The colouring in is done in either two or three different colours. The choice is made automatically so that no adjacent segments are in the same colour. The one exception is a pie of seven segments where slices one and seven are in the same colour.

Labels and Data both toggle between Inside, Outside, None and Manual. There are a number of labelling possibilities and the detailed use of these options is described in detail below.

One and two allow drive 1 and drive 2 to

```
5670     AT#0,6,8:PRINT#0,"LABELS",,Lab$
5680    = 68,100
5690     Datlab$=Dat$:datlab:Dat$=Datlab$
5700     AT#0,7,8:PRINT#0,"DATA",,Dat$
5710    = 49,33
5720     AT#0,9,24:INPUT#0,d1$
5730    = 50,64
5740     AT#0,10,24:INPUT#0,d2$
5750    END SELect
5760 END REPeat loop
5770 END DEFine
5799 :
5800 DEFine PROCedure datlab
5810 IF Datlab$<>"Outside"
5820  IF Datlab$<>"Inside "
5830   IF Datlab$="None    "
5840    Datlab$="Manual "
5850   ELSE
5860    Datlab$="Inside "
5870   END IF
5880  ELSE
5890   Datlab$="Outside"
5900  END IF
5910 ELSE
5920  Datlab$="None    "
5930 END IF
5940 END DEFine
5999 :
6000 DEFine PROCedure import
6010 CLS#4
6020 WINDOW#0, 436,24,39,11
6030 PRINT#0,"LOAD "&d2$
6040 AT#0,0,5:INPUT#0,load$
6050 IF load$="":AT#0,0,10:INPUT#0, load$:load$=d2
$&load$
6060 d2$=load$(1 TO 5)
6070 IF load$(LEN(load$)-3 TO )<>"_exp":load$=load
$&"_exp"
6080 IF load$=d2$&"_exp":CLS#0:RETurn
6090 Title$=load$(6 TO LEN(load$)-3)
6100 count=0:count1=-1:a$=""
6110 OPEN_IN#7,load$
6120 REPeat in
6130  key=CODE(INKEY$(#7))
6140  IF EOF(#7):CLOSE#7:RETurn
6150  SELect ON key
6160   = 10
6170    IF count>0:num(count)=a$
6180    seg=count:count = count+1
6190    IF count<>count1:CLOSE#7:EXIT in
6200    a$=""
6210   = 34:IF a$<>"":Label$(count)=a$:a$="":count
1=count1+1
6220   = 45 TO 122:a$=a$&CHR$(key)
6230   = 44:a$=""
6240  END SELect
6250 END REPeat in
6260 count=13:a$=""
6270 OPEN_IN#7,load$
6280 REPeat in
6290  key=CODE(INKEY$(#7))
6300  IF EOF(#7):EXIT in
6310  SELect ON key
6320   = 10:count=0
6330   = 34:IF a$<>"":Label$(count-12)=a$:a$=""
6340   = 45 TO 122:a$=a$&CHR$(key)
6350    IF count<13:num(count+1)=a$:seg=count+1
6360   = 44:count=count+1:a$=""
6370  END SELect
6380 END REPeat in
6390 CLOSE #7
```

```
6400 END DEFine
6599 :
6600 DEFine PROCedure manual
6610 WINDOW#0,260,146,129,70:CLS#0:BORDER#0,2,248
6620 AT#0,0,1:INPUT#0,"TITLE: ";Title$:CLS#0
6630 PRINT#0,\,,"LABEL",,"VALUE"
6640 FOR n=1 TO 12
6650  PRINT#0,n
6660  seg=n
6670  AT#0,n+1,16:INPUT#0,Label$(n)
6680  IF Label$(n)="":seg=n-1:RETurn
6690  AT#0,n+1,32:INPUT#0,num(n)
6700 NEXT n
6710 END DEFine
6799 :
6800 DEFine PROCedure save_pie
6810 CLS#4
6820 WINDOW#0, 436,24,39,11
6830 PRINT#0,"SAVE "&d2$
6840 AT#0,0,5:INPUT#0,save$
6850 IF save$="":AT#0,0,10:INPUT#0, save$:save$=d2
$&save$
6860 d2$=save$(1 TO 5)
6870 CLS#0
6880 IF save$=d2$:RETurn
6890 PRINT#0,"INVERSE (Y/N)?"
6900 key$=INKEY$(#0,-1)
6910 CLS#0
6920 PRINT#4,Title$
6930 IF key$=="Y":WINDOW 512,256,0,0:RECOL 7,0,0,0
,0,0,0,0
6940 SBYTES save$,131072,32768
6950 IF key$=="Y":RECOL 7,0,0,0,0,0,0,0
6960 END DEFine
6999 :
7000 DEFine PROCedure label_pie
7010 ang=ang3
7020 angle
7030 x=70+x:y=45+height+y
7040 IF ang3<PI:y=y-4:ELSE :y=y+16
7050 IF ang3>PI/2 AND ang3<3*PI/2:x=x+2:ELSE :x=x-
6
7060 IF Lab$="Inside ":lx%(n)=x:ly%(n)=y
7070 IF Dat$="Inside ":dx%(n)=x:dy%(n)=y-4
7080 IF ang3<PI:y=y+14:ELSE :y=y-20-2*height
7090 IF ang3>PI/2 AND ang3<3*PI/2:x=x-3:ELSE :x=x+
2
7100 IF Lab$="Outside":lx%(n)=x:ly%(n)=y
7110 IF Dat$="Outside":dx%(n)=x:dy%(n)=y-4
7120 END DEFine
7199 :
7200 DEFine PROCedure alter
7210 CLS#5
7220 PRINT#5,;\ "   MENU"\\" F1 HEIGHT"\\" F2 X-RA
D"\\" F3 Y-RAD"\\" F4 DESIGN"\\" F5 TITLE"
7230 REPeat loop
7240  key=CODE(INKEY$(-1))
7250  SELect ON key
7260   = 232, 72, 104:n=height:min=0:max=15:f$="H"
:ypos=3:EXIT loop
7270   = 236, 88, 120:n=xrad:min=10:max=50:f$="X":
ypos=4:EXIT loop
7280   = 240, 89, 121:n=yrad:min=5:max=35:f$="Y":y
pos=5:EXIT loop
7290   = 244, 68, 100:design:RETurn
7300   = 248, 84, 116:CLS#4:WINDOW#0,436,24,39,11:
INPUT#0,"          ";Title$:CLS#0:RETurn
7310   = 27:menu:RETurn
7320   = 81,113:STOP
7330  END SELect
```



Average gas consumption per quarter.

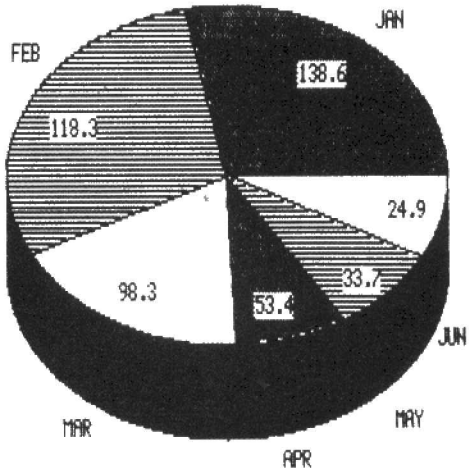
be changed. The new drive is directly inputted onto the screen. Drive 1 is used for initial loading and loading gprint_prt. Drive 2 for saving. If you wish to change the default drives this can be done by altering line 9620.

When you have completed your design changes you can leave the design menu by pressing ESC.

F5 (or "T" or "t") is for Title. When you choose this option, the present title is erased and you are invited to enter the new title. This title appears precisely in theplace that it will in the finished screen so you can place it accurately in relation to the pie.

We now return to the main menu.

F4 (or "S" or "s") is for Save. The pie is redrawn and at the top of the screen you are invited to enter the filename. The default drive is displayed and if you do not wish to alter this just press enter and then enter your filename. You are then asked if you wish to invert the image. The screen is then aved. Should you wish to use your picture in desktop publishing you should remember that the screens in these programs are smaller than the full QL screen and some experimentation or



Average gas consumption per month.

improvement with a graphics program such as *Eye-Q* may be necessary. It is also advisable to inverse the screen before saving.

F5 (or "D" or "d") is for Dump. The pie is redrawn, gprint_prt is loaded and a hard copy of the pie is made. Should you wish to use a different screen dump then change line 8480 of the program.

"Q" or "q" is for Quit. Its use in thi or in at almost any part of the program allows to leave the program.

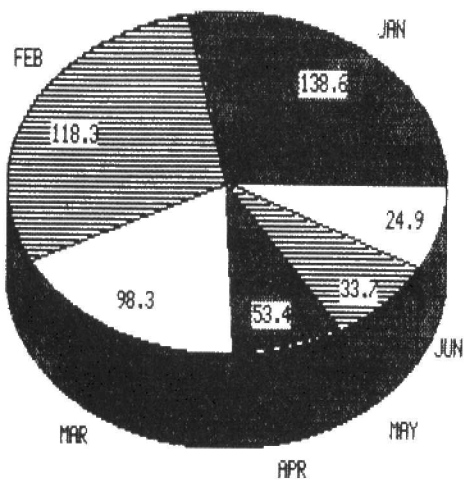
"L" or "l" or "P" or "p" is for Label or Prefix/Suffix.

"M" or "m" is for Move.

Both of these options are covered below in the section on labels.

ESC can be used almost anywhere in the program to return to the main menu. If you make a mistake and press the wrong key so that you enter either the import routine or the save routine, you can leave both routines by pressing ENTER twice and then ESC once.

There are two sorts of labels in the



Average gas consumption per month.

3Dpie. In default mode the title of a segment, label, is printed outside thepie and the value of the segment, data, is printed inside. The default options can be changed using the alter menu. The pie can also be drawn without either the data or the label displayed.

You may wish to add a prefix (eg "£") or a suffix (eg "%") to the data. This is done by pressing "L" or "l„ for Label or "P" or "p" for Prefix/Suffix. You can then enter the required characters at the top of the screen.

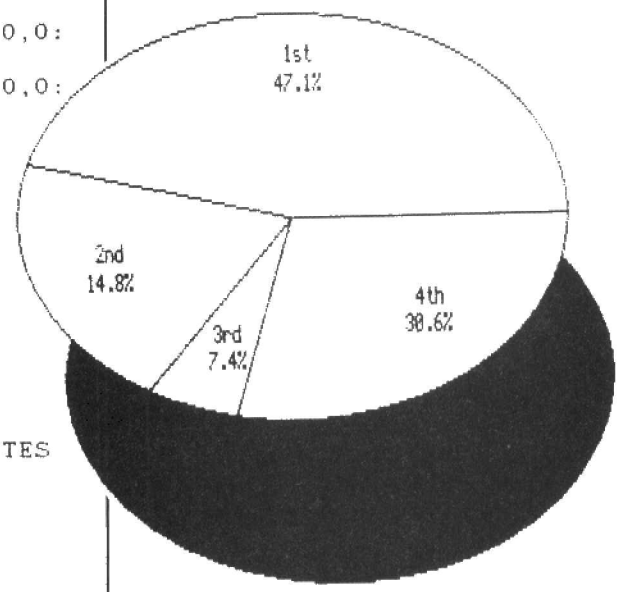
Labels

The printing positon of the labels will rarely be correct, although the computer will hazard a guess which in most cases will be approximately right. You can however change the display position: first of all press ESC to access the main menu and then F2 to draw the pie. You then press "M" or "m" to move the labels. The label of the first segment will change from white to red and can be moved around the screen using the cursor keys.

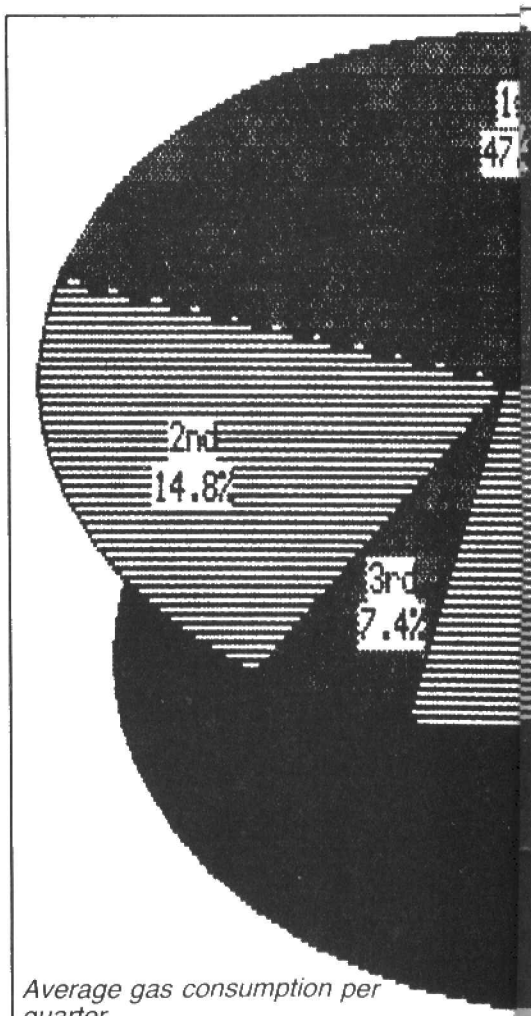
```
7340 END REPeat loop
7350 CLS#5
7360 PRINT#5;\"   ALTER"\\"   WITH"\\" ↑  AND ↓ "\\
"   THEN"\\"   PRESS"\\"   SPACE"
7370 REPeat loop
7380  key=CODE(INKEY$(-1))
7390  SELect ON key
7400   = 32:EXIT loop
7410   = 208:n=n+1
7420   = 216:n=n-1
7430  END SELect
7440  IF n>max:n=min
7450  IF n<min:n=max
7460  AT#6,ypos,8:PRINT#6, "  "
7470  AT #6,ypos,8
7480  IF f$="H"
7490   PRINT#6,2*n
7500  ELSE
7510   PRINT#6,n
7520  END IF
7530 END REPeat loop
7540 IF f$="H":height=n
7550 IF f$="X":xrad=n
7560 IF f$="Y":yrad=n
7570 view_pie
7580 REPeat loop
7590  IF CODE(INKEY$(-1))=27:menu
7600 END REPeat loop
7610 END DEFine
7799 :
7800 DEFine PROCedure data_in
7810 CLS#5
7820 PRINT#5,;\ "   MENU"\\" F1 DEMO"\\" F2 MANUAL
"\\" F3 IMPORT"
7830 REPeat loop
7840  key=CODE(INKEY$(-1))
7850  SELect ON key
7860   = 81,113:STOP
7870   = 27:menu
7880   = 232, 68, 100:initialise:demo:view_pie
7890   = 236, 77, 109:initialise:manual:view_pie
7900   = 240, 73, 105:initialise:import:view_pie
7910  END SELect
7920 END REPeat loop
7930 END DEFine
7999 :
8000 DEFine PROCedure view_pie
8010 CLS:CLS#4
8020 PRINT#4,Title$
8030 IF Title$="NO DATA LOADED":RETurn
8040 draw_pie
8050 tot=0:ang2=0
8060 FOR n=1 TO seg:tot=tot+num(n)
8070 num1=0
8080 FOR n=1 TO seg
8090  num1=num1+num(n)/tot
8100  num2=num1-num(n)/(2*tot)
8110  ang1=ang2
8120  ang2=num1*2*PI
8130  ang3=num2*2*PI
8140  IF filcol$<>"Outline":fill_pie
8150  slice_pie
8160 END FOR n
8170 IF filcol$<>"Outline"
8180  INK 7:FILL 0
8190  ELLIPSE 70,45+height,xrad,yrad/xrad,PI/2
8200  IF side$="Outline" AND "Effect$"="Solid "
8210   LINE 70+xrad,45-height TO 70+xrad,45+height
8220   LINE 70-xrad,45-height TO 70-xrad,45+height
8230  END IF
```

```
8240 END IF
8250 FOR n=1 TO seg
8260  PAPER 0:INK 7:OVER 0
8270  IF Lab$<>"None    ":CURSOR lx%(n),ly%(n),0,0:
PRINT Label$(n)
8280  IF Dat$<>"None    ":CURSOR dx%(n),dy%(n),0,0:
PRINT pre$;num(n);suf$
8290 NEXT n
8300 END DEFine
8399 :
8400 DEFine PROCedure choice
8410 REPeat loop
8420  key=CODE(INKEY$(-1))
8430  SELect ON key
8440    = 232, 73, 105:data_in
8450    = 236, 86, 118:view_pie
8460    = 240, 65, 97:alter
8470    = 244,83,115:view_pie:save_pie
8480    = 248,68,100:view_pie:a=RESPR(1400):LBYTES
d1$&"gprint_prt",a:CALL a
8490    = 27:menu
8500    = 81,113:STOP
8510    = 77,109:move_label
8520    = 76,80,108,112:presuf
8530  END SELect
8540 END REPeat loop
8550 END DEFine
8599 :
8600 DEFine PROCedure start
8610 WINDOW 454,240,31,10:BORDER 1,7
8620 OPEN #4,scr_:WINDOW#4,448,24,33,11:BORDER#4,6
8630 CSIZE#4,1,0:INK#4, 7
8640 PRINT#4, "THREE DIMENSIONAL PIE",,1989 Geoffr
ey Wicks"
8650 WINDOW 454,240,31,10:BORDER 1,7
8660 draw_pie
8670 END DEFine
8799 :
8800 DEFine PROCedure menu
8810 OPEN#5,scr_:WINDOW#5, 75,146,48,70:BORDER#5,2
,234:CLS#5
8820 CSIZE#5,0,0:INK#5,4
8830 PRINT#5,;\ "   MENU"\\" F1 INPUT"\\" F2 VIEW"
\\" F3 ALTER"\\" F4 SAVE"\\" F5 DUMP"
8840 OPEN#6,scr_:WINDOW#6,75,146,395,70:BORDER#6,2
,234:CLS#6
8850 CSIZE#6,0,0:INK#6,4
8860 PRINT#6,\"  STATUS"\\" Ht:"\" x-rad:"\" y-rad
:"\\" Label:"\" Data:"\\" D1:"\" D2:"
8870 AT#6,3,8:PRINT#6,2*height:AT#6,4,8:PRINT#6,xr
ad:AT#6,5,8:PRINT#6,yrad
8880 AT#6,7,8:PRINT#6,Lab$(1):AT#6,8,8:PRINT#6,Dat
$(1):AT#6,10,5:PRINT#6,d1$:AT#6,11,5:PRINT#6,d2$
8890 choice
8900 END DEFine
8999 :
9000 DEFine PROCedure angle
9010 ang=2*PI+ang
9020 x=SQRT(yrad^2/((1/COS(ang)^2)-1+(yrad^2/xrad^
2)))
9030 y=SQRT(xrad^2/((1/SIN(ang)^2)-1+(xrad^2/yrad^
2)))
9040 ang=ang-2*PI
9050 IF ang>PI/2 AND ang<=PI:x=-x
9060 IF ang>PI AND ang<=3*PI/2:x=-x:y=-y
9070 IF ang>3*PI/2 AND ang<=2*PI:y=-y
9080 END DEFine
9199 :
9200 DEFine PROCedure slice_pie
9210 FOR slice=1 TO 2
```

Average gas consumption per quarter.



When the label is in the correct position press space and the colour returns to white. The data now changes to red and so the process is repeated for each of the segments until the pie is completed. It is important to bear in mind that in a large pie the positon of the label can be outside the



Average gas consumption per quarter.

screen and thus cannot be seen.

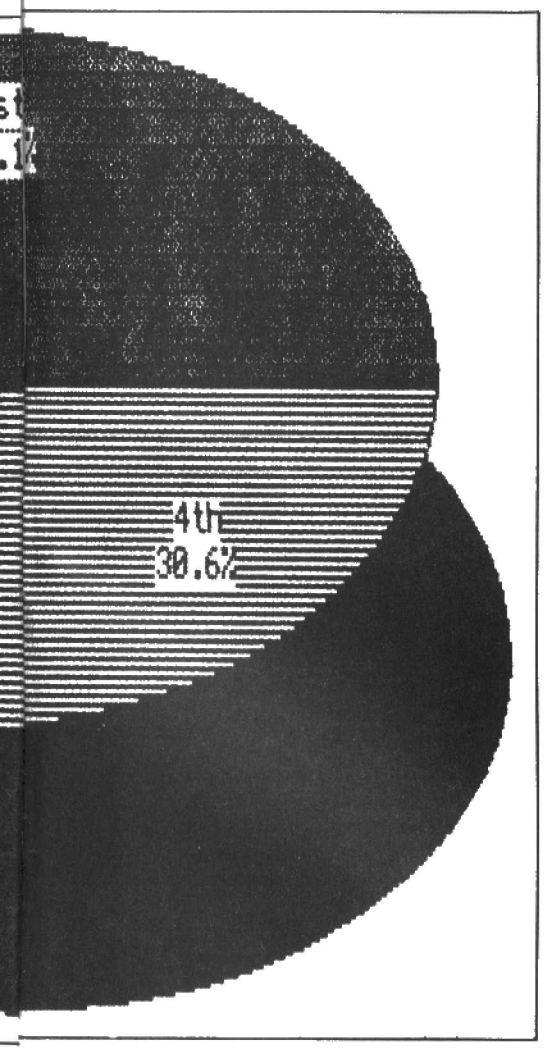
If this happens do not panic. The position co-ordinates are displayed at the top of the screen and you will see them change with the use of the cursor keys. Eventually the label will be printed in the screen area. When this routine is used the label and data options on the design menu are changed to manual and remain so until changed or until fresh data is loaded. You can leave the routine at anytime by pressing ESC

When put into writing this procedure appears unwieldly and complicated. In practice it is not, and it works smoothly and quickly.

This, in summary, is the recommended working order:

1) Load the relevant data into the programme.
2) Alter the height, x-radius and y-radius
3) Add the prefix or suffix to the data label is required.
4) Alter the approximate labelling position if a different one from the default position is required.
5) Move the labels to the exact position required.
6) Enter the colouring desired.
7) Save or printout the screen.

I hope you enjoy using the program as much as I enjoyed the challenge of writing it.




```
9220 IF slice=1:ang=ang1:ELSE :ang=ang2
9230 angle
9240 x=70+x:y=45+height+y
9250 LINE x,y TO 70,45+height
9260 IF ang>PI AND effect$="Solid ":LINE x,y TO x,
y-2*height
9270 NEXT slice
9280 label_pie
9290 END DEFine
9399 :
9400 DEFine PROCedure draw_pie
9410 IF effect$="Shadow":r=2+INT(height/5):ELSE :r
=0
9420 INK bcol:FILL 1
9430 ELLIPSE 70+2*r,45-height-r,xrad,yrad/xrad,PI/
2
9440 INK 7:FILL 0
9450 IF effect$="Solid "
9460  ELLIPSE 70,45-height,xrad,yrad/xrad,PI/2
9470  INK bcol:FILL 1
9480  LINE 70+xrad,45-height TO 70+xrad,45+height
9490  LINE 70-xrad,45-height TO 70-xrad,45+height
9500 END IF
9510 INK 0:FILL 1
9520 ELLIPSE 70,45+height,xrad,yrad/xrad,PI/2
9530 INK 7:FILL 0
9540 ELLIPSE 70,45+height,xrad,yrad/xrad,PI/2
9550 IF effect$="Solid "
9560  LINE 70+xrad,45-height TO 70+xrad,45+height
9570  LINE 70-xrad,45-height TO 70-xrad,45+height
9580 END IF
9590 END DEFine
9599 :
9600 DEFine PROCedure initialise
9610 bcol=0
9620 d1$="mdv1_":d2$="mdv2_":Lab$="Outside":Dat$="
Inside ":pre$="":suf$=""
9630 height=15:xrad=40:yrad=20
9640 Title$="NO DATA LOADED"
9650 DIM num(12):DIM Label$(12,12)
9660 DIM lx%(12):DIM ly%(12):DIM dx%(12):DIM dy%(1
2)
9670 borcol$="On ":borcol=7
9680 effect$="Solid "
9690 side$="Outline"
9700 filcol$="Outline"
9710 END DEFine
9799 :
9800 DEFine PROCedure demo
9810 RESTORE 9890
9820 READ seg,Title$
9830 FOR n=1 TO seg
9840  READ Label$(n)
9850 END FOR n
9860 FOR n=1 TO seg
9870  READ num(n)
9880 END FOR n
9890 DATA 4,"AVERAGE GAS CONSUMPTION PER QUARTER (
1984 - 1988)"
9900 DATA "1st","2nd","3rd","4th"
9910 DATA 47.1,14.8,7.4,30.6
9920 tot=num(1)+num(2)+num(3)+num(4)
9930 suf$="%"
9940 END DEFine
10000 STOP
10001 DEFine PROCedure update
10002 DELETE mdv2_3Dpie
10003 SAVE mdv2_3Dpie
10004 END DEFine
```